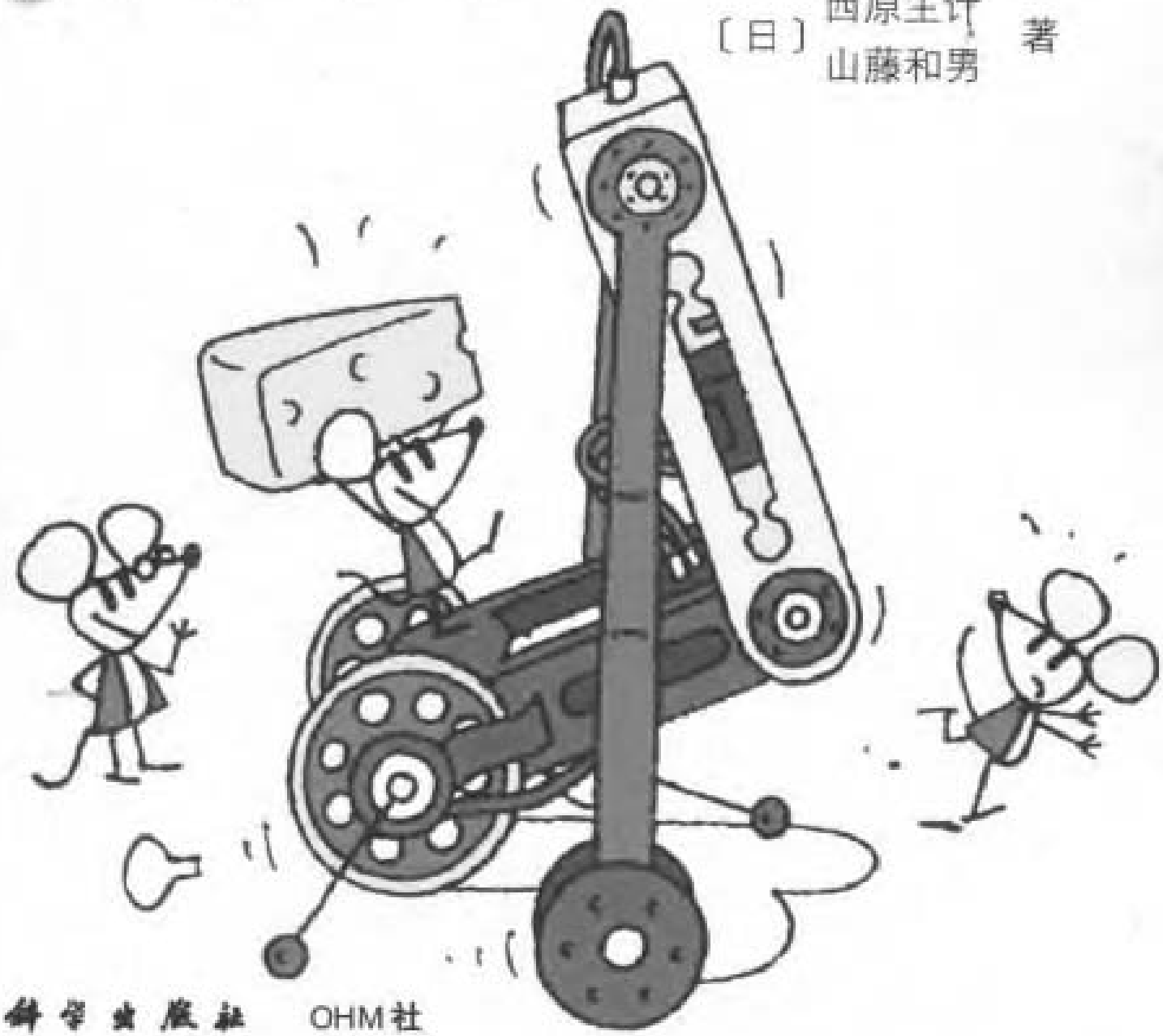
机器人竞技系列

机器人C语言
机电一体化接口

〔日〕西原主计
山藤和男 著



科学出版社 OHM社

(TP-1701.0101)
责任编辑　赵丽艳　樊友民
责任制作　魏　谨
封面制作　李　力

- 机器人C语言机电一体化接口
- 有视觉机器人制作
- 机器人竞赛指南
- 机器人制作宝典
- 机器人组装大全
- 自律型机器人制作

ISBN 7-03-010107-3
9 787030 101075 >

ISBN 7-03-010107-3/TP · 1701
定 价：30.00 元

机器人竞技系列

机器人C语言机电一体化接口

〔日〕西原主计　山藤和男　著
牛连强　赵文珍　译

科学出版社　OHM社
2002

图字:01-2002-0291 号

Original Japanese edition
C Gengo ni Yoru Jissen Mechatro Interface
by Kazue Nishihara and Kazuo Yamafuji

Published by Ohmsha, Ltd.
This Chinese language edition is co-published by Ohmsha, Ltd. and Science Press

RoboBooks
C語言による実践メカトロインタフェース
西原主計 山藤和男　オーム社　2000年　第1版第1刷

图书在版编目(CIP)数据

机器人 C 语言机电 -体化接口 /(日)西原主计,山藤和男著;牛连强,赵文珍译.
-北京:科学出版社,2002
(机器人竞技系列)
ISBN 7-03-010107 3

Ⅰ.机…　Ⅱ.①西…②山…③牛…④赵…　Ⅲ.①机电一体化-设备-接口　②C 语言 程序设计　Ⅳ.TP271

中国版本图书馆 CIP 数据核字(2002)第 08182 号

北京东方科龙图文有限公司　制作
http://www.okbook.com.cn
科学出版社　OHM社　出版
北京东黄城根北街 16 号　邮政编码:100717
http://www.sciencep.com
中国科学院印刷厂　印刷
科学出版社发行　各地新华书店经销

2002 年 3 月第　一　版　开本:B5(720×1000)
2002 年 3 月第一次印刷　印张:14 1/2
印数:1　5 000　　字数:276 000
定　价:30.00 元
(如有印装质量问题,我社负责调换(新欣))

译者序

机电一体化是一门交叉性学科，涉及机械、电子和计算机等多个学科领域，具有很强的实用性。虽然在国外，特别是在日本，对该学科的学习和研究已非常普及，但在国内的学校里尚较少开设此类课程，以此为背景的书籍也不多见。

该书作者从实践出发，较全面地介绍了有关机电一体化的内容和相关知识。作者从机电一体化的概念和外围环境入手，以实例的方式，介绍了机电一体化设备的组成、工作机理和控制方法，对目前广泛应用的大多数接口都进行了讨论，读者可以边参看内容介绍边动手实践。

本书从选材到内容介绍都以实用为主，没有过多的理论描述，大量的控制程序代码甚至可以原封不动地使用，这些对初学者或应用实践者都是很好的素材。此外，为了配合对书中所介绍知识的理解，每章都有一定数量的例题和习题，其中的习题常常含有某些应用背景并配有参考答案。

本书可以作为高等学校工科学生的参考教材和参考书，也可供从事机电一体化工作的技术人员参考。

在本书的翻译过程中，我们尽量尊重原著并保持其风格，对原文中个别不妥之处作了适当地更正并附以说明，也对原书中不太一致的程序语言代码作了调整，使其更协调。译文中的插图直接采用原图制版。

此外，在本书的翻译过程中，曾得到过沈阳工业大学的高振太高级工程师、孙宪奇副教授、王溪波副教授和邓玉昆副教授的许多帮助，田艳丰讲师也为译者提供了相关的参考资料，在此对他们表示深深的谢意。

限于译者水平，加之时间仓促，译文中肯定存在着不妥和疏漏之处，敬请专家和广大读者不吝赐教。

牛连强

前　言

机电一体化(mechatronics)一词源于日本，现在已在全球范围内使用，是机械学(mechanics)和电子学(electronics)两词的合成词。作为一门学科，机电一体化融合了机械学、电子学和计算机3个学科领域的知识，形成了所谓智能信息机械科学与技术。其中，计算机的地位举足轻重。实际上，大量的智能机械，如VTR(磁带录像机)、照相机和汽车等，都是通过微型计算机控制的，只是往往不被人们察觉而已。

所谓接口，是指实现两事物之间沟通的媒介。例如，一个只会讲本国语言的人，在与外国人进行语言交流时需要翻译，此翻译即相当于接口。当事物间存在着不同的现象和思考方法以及不同的形态和数量(包括模拟量和数字量)时，使其能够相互关联的媒介物即是接口。例如，数字计算机和模拟传感器之间的接口是AD转换器；人和计算机之间的接口包括键盘、鼠标器、扫描仪和显示器等等；而遥控器通常则是人和家用电器之间的一种接口。

在设计和控制不同的机电一体化设备时，必须对机械结构、执行装置、传感器和计算机等进行认真地研究和探讨，而执行装置、传感器和计算机之间的硬件及软件连接技术统称为机电一体化接口。机械和计算机、机械和机械、人和机械等之间的信息和意图通过接口才得以沟通。

微处理器是整个系统的控制和通信中心。不过，只有通信网络、键盘和鼠标器所构成的系统还是无法运行的，必须使用具有中介作用的接口(检测控制板或称接口卡)。此外，为了能够借助接口板建立起传感器和执行装置之间的连接，我们还必须编制相应的存取控制程序。

对于学习检测控制技术的学生来说，机械与计算机相结合的机电一体化接口技术是必须掌握的实用技术。尽管有关机电一体化的结构和运动控制方面的入门书籍已为数不少，但以电路构成和计算机程序等实例为主题的教材却不多见。很长一段时间，笔者一直考虑通过工程技术书籍的形式，将精心收集的一些实例展示出来，使读者可以了解如何去掌握并实现这些技术。

由于笔者长期使用 PC-9800 系列(NEC 公司)产品，故本书中采用的背景语言是 MS-C(Microsoft C)ver. 6。不过，因为近来的应用研究都集中在 DOS/V 机上，因此，也将计算机外部设备的控制尽可能用 C/C++来描述，目的是使原来的内容可以移到 MS-DOS 平台的 MS-VC(Microsoft C/C++ ver. 8. 03)和 Turbo C++ ver. 4 for DOS 上。

事实上，即使在 Windows 操作系统中，如果切换到 DOS 模式(可称为 DOS 窗口)，大多数计算和显示用的 C/C++程序都可以正常执行。执行"exit"命令可使系统从 DOS 窗口返回到 Windows。但如果在控制某些接口板时需要频繁地进行切换，则最好通过程序控制来完成这些启动和退出的动作。应该说，在进行检测控制时，操作系统(OS)最好采用完全的 MS-DOS。关于不同的 OS 的区别及对程序的影响，本书也做了较清楚的描述。

本书内容以机械系统工程学专业的本科 3 年级讲义为基础，并以读者具有数字电路和 C 语言的基础知识为前提编写而成。书中所涉及的知识皆以尽量简短的条目形式列出，可以作为毕业设计的基本材料参考使用。从形式上看，本书较以前的书籍更容易阅读。此外，书中各章的例题和习题，尽可能给出了答案以便于读者自学。鉴于接口技术的重要性，建议读者能够一边认真学习理论，一边动手实践，只有这样才能真正掌握这门技术。自学时，对于那些易读的示例程序和注释也应认真调试，以加深体会。此外，DOS/V 机根据机种不同其 BIOS 设置也有差异，错误的设置可能导致不同的结果，出现错误时可以尝试自行修正。

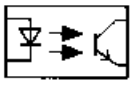
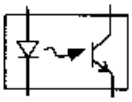
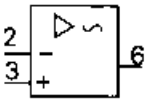
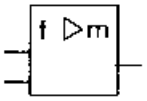
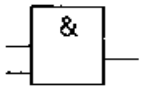
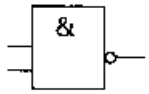
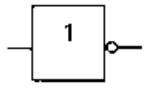
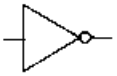
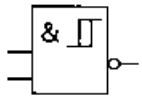
学习完本书的内容后，可以继续阅读参考文献中所列出的专业性书籍，以便加深理解。随之，可以在毕业设计和实际工作中进一步学习有关控制工程、传感器工程、机电一体化和机器人等相关领域的知识，使机电一体化真正成为自己的一项能够灵活运用的基本技能。

著　者

关于新的“JIS 电气图用图形符号”

近年来，日本工业标准 JIS 已逐渐全部过渡到国际标准化组织 ISO(与电气相关的组织是 IEC)的国际标准。本书中的参考图示，是根据日本标准协会在 1999 年 9 月发布的最新“JIS 电气图用图形符号”，并剔除了难以理解的部分而绘制的。主要的一些电气图形符号变化可参见如下的新旧图形对照表。

新旧 JIS 电气图用图形符号对照表

部 件	新JIS	原符号	备 注
电阻			
线圈			CAD不支持原来的符号
晶体管(NPN)			•为集电极，⊢→为N发射极
晶体管(PNP)			⊢←为P发射极
光电二极管			为可见光
LED			连续发射
稳压二极管			
光电耦合器			
OP放大器			三角表示传送方向
放大器			m为放大倍数 f:Σ累加，∫ 积分
与门			
与非门			用≥1代替&即为NOR
反相器			缓冲器去掉○
施密特触发电路			NAND型
单向总线			
双向总线			

目　录

Chapter 9 执行装置接口

附　录

chapter 1

机电一体化接口技术基础

为了实现对机电一体化设备的设计和控制，必须对机械结构、执行装置、传感器和计算机等进行研究和探讨。接口是指用在执行装置、传感器与计算机之间，使其信息和指令得以沟通和交换的装置。本章仅从接口的角度介绍机电一体化设备的相关内容，目的是加深对机电一体化接口概念的理解。

1.1 机电一体化设备的组成

1.1.1 接 口

1. 机电一体化

这是指以微型计算机(微处理器)为中心而组成的电子控制机械。随着微处理器在小型化、高速化、低价格及节约能源等方面的发展,将其与机械设备相结合而形成的机电一体化已成为当今的极为兴旺的产业。不过,计算机本身并不能直接与机械设备进行连接。

2. 接 口

对于机械和机械的连接点,通常可单纯地称其为接口。在对图 1.1 所示的自动化设备进行控制时,需要添加相应的**外部设备**,我们将这些设备划分为机械和计算机的接口以及计算机和人的接口来描述。为了实现机械部分、信息处理部分同计算机的会话,作为中间媒介的外部和外部设备(此处是 AD 转换器和 DA 转换器)以及检测控制软件是不可缺少的,通过这些设备才能实现信号的读取和交换。

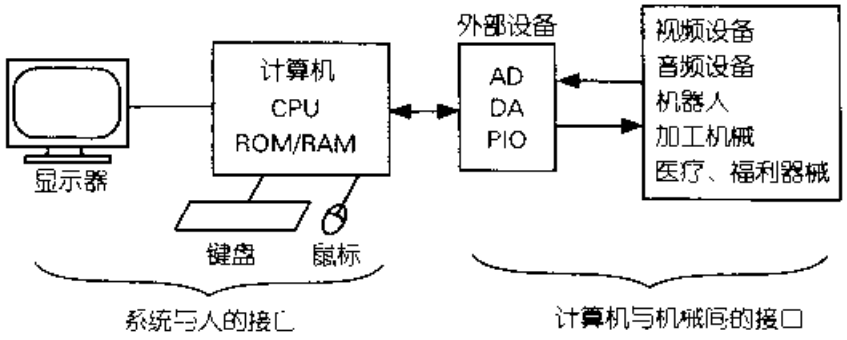


图 1.1 接口设备

事实上,用于连接信号的引脚或连接器,以及开关之类的装置都是接口。近来,由于很多接口部分都允许用户根据自己的需要进行设置,因此,在设计时应考虑适当采取一些自动纠错的辅助配置。

3. 接口板

通常,控制板(或称控制卡、接口卡及接口板)要插在计算机内部或者后面的插槽(可称为扩展槽)中,且每个接口都由一块电路板(有时也称其为基板,上

面附有印刷线路、芯片等)构成。最初,很多应用中都需要自己设计并制作这样的接口板,但因最近商品板的价格逐渐便宜,种类也较多,直接选购商品板的人也日渐增加。电机驱动器(驱动电路)、电源等提供能源的设备,应特别注意,需要认真挑选和准备。当然,仅仅靠这些硬件设备并不会使机械系统工作,我们还必须在外部设备中加上简单电路,连接好导线并编制计算机控制程序,才能形成一套真正意义上的机电一体化设备。这些必须自己了解并实现的工作就集中在**接口技术**部分。

这里所说的**主要外部设备**包括:

(1) 开关控制盒——用自动或手动方式控制器械处理过程的遥控器等

(2) 并行 I/O——多位开关量的并行输入输出设备

(3) AD 转换器——模拟量 / 数字量转换器

(1) DA 转换器——数字量 / 模拟量转换器

(5) 定时器计数器——每隔固定时间输出一个基准脉冲,并对脉冲进行计数的设备

(6) 编码板——对编码脉冲进行计数,也称为计数板或计数盘

(7) 电机驱动器——电机等的驱动电路(在进行机械驱动时必须使用电流放大器)

(8) 声控板——具有 FFT(快速傅里叶变换)功能的 DSP(数字信号处理)板

(9) 图像板——从照相机读取图像数据

(10) 通信板——用于实现检测数据的传输和控制指令的发送

1.1.2 人机工程接口

设备、机械、构件和环境等与人接触的界面称为人机工程接口。图 1.1 中所示的键盘、鼠标和显示器等即属于此类接口。例如,适应手指动作节奏的键盘、对眼睛无损伤的显示器、不易使人疲劳的椅子和桌子以及紧急情况的应急标识等等,都会在使用心理和安全性等方面对使用者产生深刻的影响。

室内空间的温度和湿度、空间的大小和色彩搭配、门槛的高低、建筑物的外观、街面道路和环境、服饰、匙和杯子的把手、各种标记乃至材料表面的光泽等等,是人和物体、人和环境的接触点,都属于人机工程接口。此外,移动装置用声音发出的警告、汽车中系好安全带的警告、具有福利性质的机电一体化设备(指残疾人用品)给出的介绍和引导信号(包括姿势、声音和图像等),以及模拟现实环境等都是人机工程接口。对于这样的接口,在设计时应该考虑具有亲切感、温馨感和美感等因素。

1.1.3 人机接口

这是指机械设备中由人操纵的部分。飞机的操作盘、机器人的示教盒、收音机的面板等都是这方面的典型实例，这种接口重点体现机械设备的可操作性和安全性。例如，把手、开关和警示灯等，是按照人的习惯来设计的，而对于紧急情况的设计原则，应以操作员可以在最短时间内采取制动措施为准。因此，考虑紧急停止按扭时，外观不是主要依据，一般采用大的红色按钮，使其容易看到，并应放置在人的反射动作容易达到的位置。此时，不必过分考虑气氛是否和缓、融洽。

应该注意到人与机器的接口与人机工程接口是不同的，例如，汽车驾驶员座位周围的方向盘、离合器和闸等都是人与机器的接口，座位、道路标记等反映人和物、**人和环境的关系**，从此意义上说，是人机工程接口。

1.1.4 机器人的接口

1. 伺服系统

伺服(servo)意味着仆从(servant)，即忠实地遵从并执行主人(master)的指令和意图，此处的主人是指人或计算机。伺服系统是反馈控制的典型例子，在飞机和船舶的舵轮装置、雷达跟踪、NC 机床及机器人等工业设备中使用极为广泛。这里我们以机器人为例，简要说明伺服系统的 3 个主要组成部分，参见图 1.2[1]。

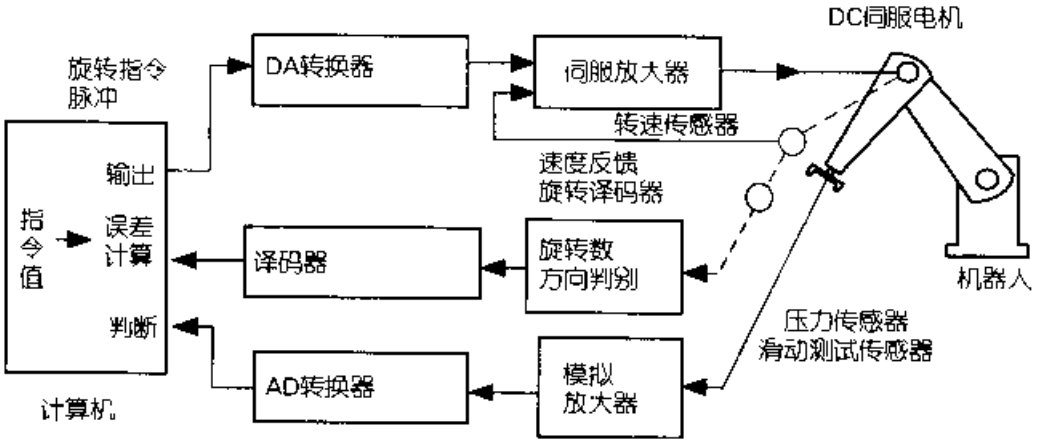


图 1.2 机器人的伺服控制机构

(1) 执行装置部分——由伺服驱动器、DC 电机、AC 电机等构成。伺服驱动器能够将电压信号进行电流放大，这在电机和扬声器等的电流驱动型的机械

控制中是不可缺少的。

(2) 控制部分——由计算机和位置、速度以及加速度的控制部分构成。这些设备统称为控制器。

(3) 传感器部分——由检测转速的转速传感器(TG)、检测转角的旋转编码器和旋转方向判别电路及压力传感器等构成。转速传感器作为伺服放大器的速度控制部分直接反馈,而检测压力、接触和滑动信息的抓取传感器所测得的信号,经 AD 转换器转换后再输入到计算机。

2. 位置控制

为了实现位置控制,计算机需要先计算出指令值和旋转编码器的反馈值之差,并输出给 DA 转换器。DA 信号经过伺服放大器放大后驱动 DC 电机转动,新的旋转角信息从旋转编码器再次反馈到计算机。根据这种循环可以达到误差信号为 0 的反馈控制。

3. 外表看不见的接口

如前所述,AD 转换器和 DA 转换器是处理模拟量和数字量的接口,不是暴露在外的独立装置。AD 和 DA 转换器的应用极为广泛,不知您是否意识到,即使是移动电话也要使用 AD 和 DA 转换器。

1.1.5 伺服系统的结构

1. 最简单的伺服机构

如前所述,伺服机构是指能将人的意图迅速准确地实现的装置。图 1.3 是位置控制的模拟伺服机构示意图。图中,输入轴和输出轴的角度,是由不同电位器测得的电压值来体现,它们的差信号经过伺服放大器的放大,驱动电机工作。当差信号变为 0 时电机即停止运转。以很小的力使输入轴旋转,就可以在输出轴上产生较大的旋转力矩,故可用于操纵大型机械。

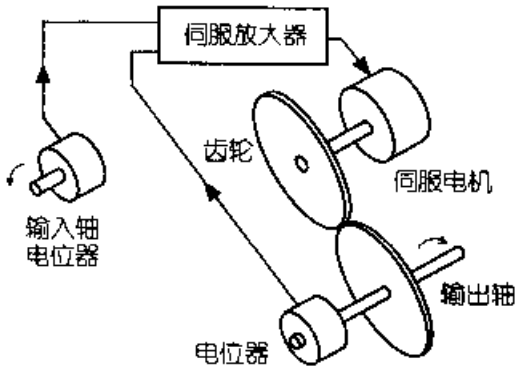


图 1.3 位置伺服机构

2. 姿势控制

在日常生活中，为了克服重力，人们常以不同的姿势站立。此时，人体中的重力反馈控制系统会自动启动并工作。其中，最基本的部分包括，脚的肌肉传感器、韧带传感器、脚底压力传感器和脊椎中枢的自动反馈控制系统。作为一个简单的例子，可参考如图 1.4 所示的平行二轮车机器人。这是一个利用控制臂保持平衡、通过转动车轮进行移动的移动机器人。控制臂和车体的合成重心位置的改变控制着姿势，车轮只是控制行走，不参与其它控制工作。

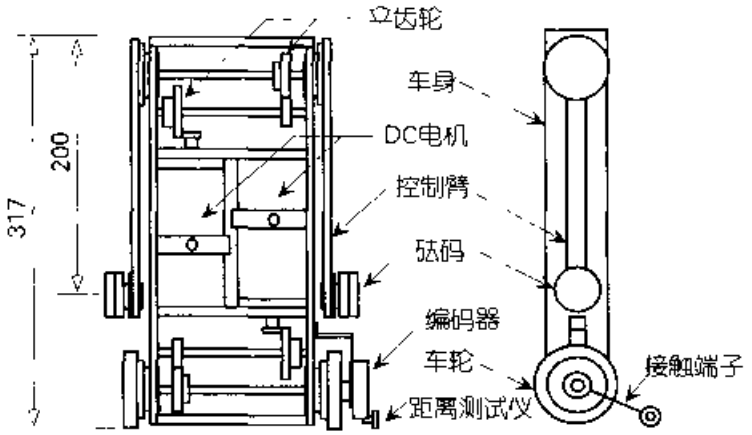


图 1.4 悬挂摆臂式平行二轮车机器人[50]

以下是对此系统的简单描述：

(1) 结构——高度约 32cm，重量约 5kg，顶端和底部各使用一个 60W 的 DC 电机，臂的前端设有保持平衡用的砝码。

(2) 传感器——采用 500 脉冲/转的电机轴编码器。为了检测车体的倾角，在车轮电机轴上设置一个旋转编码器(2048 脉冲/转)，并从轴上向地面引出触杆。此外，触杆的前端装有车轮，用旋转编码器(100 脉冲/转)即可测量出移动的距离。

(3) 控制系统——用计算机和 C 语言编制的程序进行控制。每次采样可以将传感器信号送入计算机，通过运算，生成电机的转矩指令。该指令经 DA 转换器输出到电机驱动器，使电机产生转矩并控制各轴的转动。然后，角度信息会再次被送入计算机，进入下一次的循环。

1.1.6 家庭用的智能机电一体化设备

所谓“智能”，主要是指“对自身的运行状态可以进行适度判断”的功能，这是通过传感器技术和控制技术的进步而发展起来的。事实上，家用电器中大量

使用传感器，并借助于计算机芯片进行状态判断。

1 微波炉

微波炉是通过电磁激励的微波经食品吸收，使食品自身发热，进而达到烹调食品的目的。不过，根据被烹调食品的种类和数量不同，消耗的电量和所花的时间也有差异。借助传感器设备，微处理器可以实现对烹调过程的自动控制。此处所使用的传感器主要包括温度传感器和湿度传感器。

食品表面放射的红外线可先由温度传感器接收到，经过 AD 转换器将信号数字化，再与微处理器内存中事先存储的烹调数据相比较，并根据比较的结果控制电磁强度。

2. VTR

VTR 是一种机电一体化程度很高的设备，特别是由于旋转磁鼓和磁带运转部分最为关键，因此，使用了包括磁性、温度、光、结露等多种类型的传感器。图 1.5 是采用 DD(直接驱动)电机的磁带读取部分的伺服结构示例[2]。

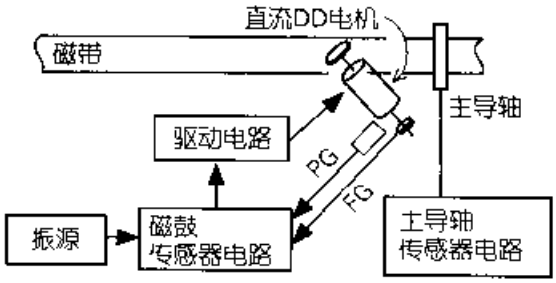


图 1.5 VTR 磁带读取部分的伺服机构

(1) 磁带读取部分——用旋转磁鼓 FG 传感器(霍尔 IC)控制转数，用 PG 传感器(光电译码器)控制相位。

(2) 磁带运转部分——运转部分(主动轮)将 FG、PG 传感器和磁带上记录的控制信号进行对比，可以实现微米级的精确控制。

(3) 磁带终端——用光传感器可以感知磁带上的透明部分，也可以感知磁带是否松弛。

(4) 运转计数——用磁铁和磁性传感器(霍尔 IC 或磁阻元件)进行编码计数。

(5) 磁性信息——用磁性传感器(孔式 IC 或磁阻元件)实现感应测量。

(6) 结露传感器——当气温变化时，室内外环境的变化会造成结露现象。VTR 具有结露传感器和加热器，可以消除结露并能防止磁带加热时发生卷取等损伤现象。

1.1.7 TV 等的遥控器

TV、VTR、立体声收音机和空调器等设备所使用的遥控器是以**红外线方式**工作的，可以在远距离处控制电源的开关、频道的切换（包括冷热切换）及音量调节（或温度调节）等。遥控器与计算机的鼠标器和游戏机的操纵杆等一样，都是人机工程接口的媒介[3]。

（1）在遥控器端，利用 GaAs（砷化镓）的 LED（Light Emitting Diode，发光二极管）作为发光元件，可以发出 30k～60kHz 的载波。是使这种元件所发出的光区别于荧光灯、太阳光及其它红外线不会混淆。

（2）在接收信号的机器端，采用具有较高响应速度的硅光电二极管（PIN－PD）作为接收元件，再利用红外线过滤器和微处理器对符号进行解释（称为译码），不会产生误操作。

（3）一种型号的遥控器可以控制和操纵若干台设备，11 位编码中仅用 5 位与接收信号端的机种相对应[4]。

（4）红外线遥控器并不只限于对家用电器的远距离操纵，自行研制的机器人和机电一体化设备也可以通过遥控器从远距离发送指令进行控制，使用非常方便。

1.2 计算机和接口

目前，计算机的使用已极为广泛，它可以代替人类去执行很多日常或特定的工作。事实上，今天的计算机已成了人类的伙伴。对于设计者来说，需要先对计算机系统的组成有足够的了解，才能对其进行适当地控制并使其按照自己的意愿去工作。

1.2.1 关于计算机

在计算机世界中，既有超级计算机、并行计算机等超大型和超高速计算机，也有单片微型计算机（如微处理器和袖珍计算机等），可谓是种类繁多。机电一体化中主要使用的是个人计算机（PC：Personal computer）和微处理器。虽然微处理器在处理速度、通信能力方面稍差，但体积小，价格便宜，更适合在机电一体化设备的控制中使用。图 1.6 所示为计算机的一般结构，内存和相应的输入输出设备（I/O：Input/Output Unit）通过总线相连。

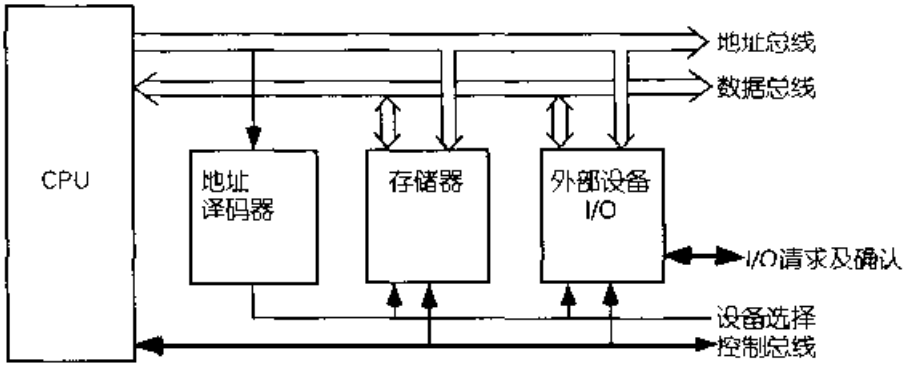


图 1.6 计算机总线的构成

1. CPU

CPU(Central Processing Unit)即中央控制器或称中央处理单元,主要由若干个寄存器、程序计数器和运算器等组成[5]。

2. 内 存

内存的种类很多,包括:

(1) ROM(Read Only Memory)——只读存储器。

(2) RAM(Random Access Memory)——随机读写存储器。

(3) EPROM——紫外线可擦除 ROM,我们可以利用 ROM Writer 对其进行程序设置。

(4) SRAM——通过电池供电即可保存数据,不需要对数据进行重复刷新。由于构成 1 位的容量需要使用 4 个晶体管,因此,这种存储器容量小且价格高,但存取速度快。通常在一些特殊的计时器和移动电话上使用较多。

DRAM(Dynamic Random Access Memory)是动态随机存储器,由于构成 1 位的容量只需要 1 个晶体管,因此,这种存储器价格便宜且容易构成较大容量的存储体。但 DRAM 没有数据保持能力,需要定时对数据进行刷新。

(5) 高速缓冲存储器(Cache)——通常利用 SRAM 存储器构成。CPU 可一次性将数据复制到 Cache 上,使得以后对数据的存取操作可以在 Cache 中进行,而不需要在主存中进行(直到 Cache 中没有所要的数据才会再次写入新的数据),从而提高了处理速度。

(6) SDRAM(Synchronous DRAM)——同步动态随机存储器,这种存储器利用同步型 DRAM 与系统时钟同步的方式进行数据存取。SDRAM 存储器的数据存取速度很快,可望成为今后计算机内存的主流产品。

3. I/O

为输入/输出接口部分,包括键盘、CRT 显示器、鼠标器、打印机、扫描仪以及 CCD 照相机等。

1. 总 线

计算机中有三种总线：

(1) 地址总线——是选择内存和 I/O 设备的信号线。

(2) 数据总线——用于接收和发送指令及数据信号。

(3) 控制总线——用于接收和发送计算机与外部设备之间的中断请求。

1.2.2 有关 DOS/V 机

1. DOS/V 机

一般我们称包括 IBM PC/AT 在内的兼容机为"DOS/V 机"，操作系统以 DOS 为基础，且通常在 PC-DOS 上附加了相应的日文字体的处理功能(国内使用的系统通常附加中文操作系统，使其具有相应的中文字体处理功能，以下同：译者注)。DOS/V 机种类繁多，如 IBM、COMPAC、DELL、富士通 FMV 和夏普等都属此类。

DOS(Disk Operating System，磁盘操作系统)基本上是指 Microsoft 公司开发的操作系统即 **MS-DOS**。"V"是指 640×480 点阵的图形显示标准 **VGA** (Video Graphics Array)，在英语版的 DOS 上附以本国语言处理软件。DOS 操作系统的最新版本包括 MS-DOS/V ver.6.2 和 IBM 的 PC-DOS/V ver.7.0。在美国把它们统称为 DOS。

在启动 Windows 98 时，先要启动 DOS 并进行环境设置。Windows 是 32 位的操作系统，兼容以前的 16 位操作系统。

2. DOS/V 的特点

PC-9800 系列(由 NEC 公司研制开发的微机系列)中固化了日文字体，而 DOS/V 中事先固化的是英文字体，但也能以图形方式显示日语。应该说，由于价格便宜，加之可以使用各种语言的软件并可运行 Windows 等诸多原因导致了 DOS/V 的普及。

在 DOS/V 机中，FDD(软盘驱动器)通常分为 A 和 B 驱动器，HDD(硬盘驱动器)主要是指 C(或 D、E 等)驱动器。

3. 内存管理

在 CPU 为 80386 以前的 16 位微机中，CPU 可以直接寻址的内存空间是 2^{16} = 64kB，除此以外的内存地址需要按段基址和偏移量的方式进行管理，MS-DOS 至今仍采用这种内存管理方式。

Windows 作为 32 位的操作系统，可以直接寻找的地址多达 2^{32} = 4GB。它将整个内存统一管理，不再采用内存分段方式。

1. DOS/V 机和 PC-9800 系列的键盘操作

DOS/V 机和 PC-9800 系列有很多不同点，表 1.1 表示了在二者的键盘操作之间的差异。

表 1.1 DOS/V 机和 PC-9800 系列机的键盘操作差异

操作	DOS/V 机	PC-9800 系列
菜单功能	Alt	GRPH
Reset	Ctrl + Alt + Del	RESET 开关
汉字输入	Alt + 半角/全角	CTRL + XFER
假名/英文	备用键	XFER
大小写转换	Shift + Caps Lock	CAPS
终止	Ctrl + C, Ctrl + Pause	STOP
拷屏	Print Screen	COPY

1.2.3 硬 盘[6]

HD(hard disk，硬盘)是微机的重要存储器，Windows 操作系统和字处理器等软件都需要安装在硬盘上。

1. IDE 标准

IDE(Integrated Device Electronics)是 DOS/V 专用的内置硬盘连接标准，具有价格低、速度快且容易扩充等特点。连接电缆的长度限定为 60cm，也可使用内置设备连接线。不过，若 1 个硬盘的容量为 504MB，此标准至多允许连接 2 个这样的硬盘。

2. EIDE 标准

Enhanced IDE 是增强型的硬盘连接标准，最多可以连接 4 个每块容量为 8.4GB 的硬盘。目前的 CD-ROM、MO(光磁盘)和 HDD 都按 EIDE 标准连接。PC 9800 系列中也使用此标准。

3. SCSI 标准

该标准利用专用的 SCSI(Small Computer System Inteface)控制卡，可以实现 HD、CD-ROM、MO、打印机和扫描仪等所有外部设备的连接，但价格较高。1 块接口卡最多可以连接 7 台外部设备。使用该卡进行设备连接时，连接线全长最多为 6m。使用 SCSI 需要进行终端设置，且最新的产品支持以开关方式进行终端选择。

此外，目前正普及使用的是 SCSI-2，且 SCSI-3(Ultra SCSI)也已标准化，并开始投入使用。

1.2.4 扩展总线

1. C 总线

这是 PC-9800 系列的扩展总线，在目前的 PC-9800 系列中使用很普遍，但因外围设备较多，数据传送速度慢，故已逐渐被淘汰。

2. ISA 总线

ISA(Industry Standard Architecture bus)是工业标准结构总线的简称，是 IBM DOS/V 机的标准扩展总线，从 1984 年开始大量销售这种总线的外围控制板。虽然使用 ISA 总线的控制板价格较为便宜，但数据传输速度比较慢，只有 8MB/s。8 位的总线称为 XT 总线，而 16 位的总线称为 **AT** 总线，PC/AT 的名称也由此而来。现在 ISA 总线的数据总线已拓宽到了 16 位，地址总线要求是 24 位。

EISA 是将 ISA 扩充到 32 位而形成的总线，通常只用于服务器。

3. PCI 总线

PCI(Peripheral Component Interconnect bus)是外部配件互连总线的简称。该总线具有 32 位乃至 64 位的数据传输能力，可见其传输速率较高，总线时钟都在 33MHz 以上。标准的 DOS/V 机、PC-9800 系列机及 Mac 机都支持这种总线，且最近的其它类型的扩展总线也都逐渐向 PCI 总线过渡。大体上说，PCI 总线具有如下一些特点：

(1) PNP(Plug & Play，即插即用)，增加控制卡时可由系统自动进行设置。

(2) 若干个设备可以共享 1 个中断请求 **IRQ**(硬件中断)。

(3) 经由 PCI 总线传输的数据块，不再采取以前的 DMA 控制器方式，而是由外部设备来控制总线(所谓的 Bus Master，即总线主控器方式)，直接对主存进行数据存取，提高了数据的存取速度。

4. USB

USB(Universal Serial Bus)称作通用串行总线，它是为了使计算机的外部设备具有统一的接口而发展起来的通用总线。此类总线在 Windows 98 系统中使用日益广泛。按数据传输速度可将其划分为 2 类，包括 12Mbps 的高速 USB 和 1.5Mbps 的低速 USB。

1.2.5 并行与串行

1. 并行接口

计算机与打印机、外部存储器以及扫描仪等外部设备之间的数据交换量是很大的，通常采用多位数据同时传送的并行接口(SCSI 等)进行数据传输，而计算机之间的通信，通常使用 RS-232C 等串行通信方式。

2. 串行接口

串行是指将字节中的每位数据串行化后再逐位传送。从技术上讲，串行方法比并行方法，更容易实现高速化的远距离数据传输。串行中还包括同步和异步两种传输方式。同步传输方式是指将数据直接串行化并按原样传输的串行方式。若采取异步方式传输，需要先将数据串行化并附加上起始位和终止位，然后再进行传输，目的是使接收方也容易达到"同步"。

练习 1

问题 1.1 请说明电机传感器系统的反馈控制结构的组成。

问题 1.2 什么是机电一体化？请给出 3 个家用机电一体化产品示例，并说明这些产品中哪些组成部分与机电一体化有关。

问题 1.3 为了对自行设计的机器人进行控制，需要使用什么样的接口板？请说明接口板的名称和使用方法。

问题 1.4 请选择与以下设计相吻合的电气电子器件。

(1) 想制作精度是 1%的差动放大器，应该选择什么样的电阻为好？
(2) 电源电路的旁路电容。
(3) TTL IC 的旁路电容。
(4) 积分电路的电容。
(5) 步进电机的电阻。
(6) 电路板上的小电位器。
(7) 稳压电路

问题 1.5 试给出一个利用硬件方法防止误操作的例子。

问题 1.6 请谈一谈在机电一体化接口方面，今后需要努力研究的问题。

chapter 2

微机和外部设备

Intel公司在1971年开发出了4位的微处理器4004，1975年Texas Instruments公司也开发出了自己的单片微处理器。此后，微处理器的应用领域不断拓宽，逐年增加，在生产自动化（FA：Factory Automation）、家用电器、游戏机、办公自动化、汽车和移动电话等生产领域已逐渐普及。利用微处理器可以构成具有智能判断和调节能力的控制系统，而目前控制领域中采用的微处理器大都是单片微处理器。本章将简要讨论微处理器系统的构成以及操纵它的汇编语言。

2.1 微处理器硬件

2.1.1 微处理器

1. 微处理器电路

微处理器电路包括 CPU(Central Processing Unit,中央处理单元)、内存以及提供输入输出接口的数字电路。其中,数字电路中表现为下述的 2 种逻辑:

(1) 硬件逻辑——用数字集成电路(IC)组成的逻辑电路直接实现机械控制。

(2) 软件逻辑——通过微处理器进行控制,使同一个电路在程序的作用下实现不同的功能。

微处理器有 4 位、8 位、16 位和 32 位之分,其产品包括单片机和单板机等多种类型。体积上,有在笔记本大小的电路板上设置 10 位数字按键形式的微处理器,也有在主电路板上集成了检测控制用的外部电路和只有手掌大小的微处理器。图 2.1 是经过扩大后的微处理器的接口部分,应注意每个外部设备与 CPU 之间都需要经过 AD 转换,输入输出端口就是我们身边的各种控制、调节装置与计算机相结合的设备。

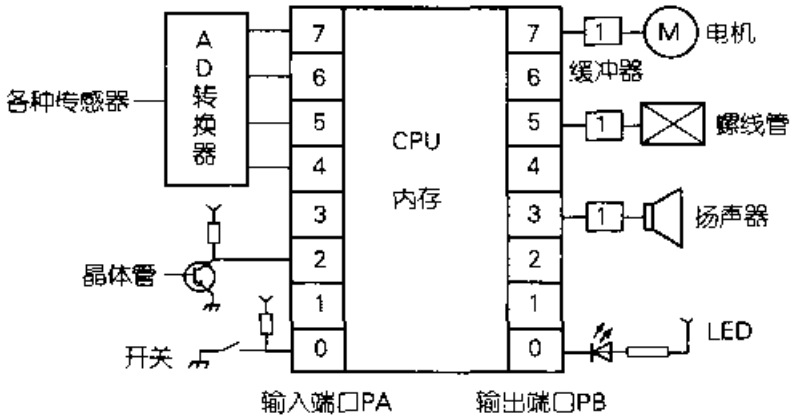


图 2.1 微处理器与控制设备的连接[3]

微处理器内部所能识别和处理的语言是机器语言,即仅包含 0 和 1 的逻辑数据,通过这些数字组合形成数值以及字符。虽然可以在计算机中使用与人类的自然语言接近的高级语言,但微处理器本身是不能直接处理这些语言的。由

于很多微处理器的程序都固化在 LSI 中，不容易更改，从此意义上也可将其看作更高级的数字电路。

2. 使用机器语言的微处理器

(1) 单片型(4 位、8 位)——用于家用电器和电子玩具等，其程序不易改变。

(2) 单板型(8 位、16 位)——用于自动售货机、交通信号机和小型运动机器人等，允许修改其控制程序。

(3) 控制用计算机(16 位、32 位)——用于工业机器人、NC 机床及大楼内的监视系统等。

3. 使用高级语言的个人计算机

(1) 通用个人计算机(16 位、32 位、64 位)——用于科学计算、图像处理、事务管理及游戏机等。

(2) 文字处理机——实现文字的录入(打字)、编辑、排版和印刷等处理工作。

(3) 袖珍计算机、工作站(32 位、64 位)——用于 LA(试验室自动化)及 CAE(计算机辅助教育)。

4. 80 系列(Intel 公司研制开发的产品)和 68 系列(Motorola 公司研制开发的产品)微处理器

(1) CISC(Complex Instruction Set Computer)——复合指令集计算机。内部含微程序，以顺序方式自动执行指令，是工作站的组成部分。

(2) RISC(Reduced Instruction Set Computer)——简明指令集计算机。RISC 以并行方式工作，可以进行高速的信息处理。与 CISC 相比，RISC 适用于执行单纯的处理任务。

2.1.2 Z-80 CPU

1973 年 Intel 公司发表了 8080 CPU，1976 年 Zilog 公司开发出了 Z-80(内嵌 8080 系统)CPU。时至今日，Z-80 仍是微处理器的主流产品之一。与 Z-80 相当的还有 Motorola 公司的产品 MC6800 微处理器。其后，Intel 的 8080 于 1978 年改为 8086，它是 NEC 公司的 PC-9800 系列的基础。

1. 请　求

在 Z-80 的主要接口功能中，来自外部设备发送给 CPU 的请求包括：

(1) INT——从 I/O 设备到 CPU 的中断请求信号

(2) $\overline{\text{BUSRQ}}$——当 I/O 设备希望进行数据存取时，要求 CPU 把总线置于高阻抗状态，并脱离总线控制权的请求

CPU 对外部设备的请求与应答包括：

(1) $\overline{MREQ}$、$\overline{IORQ}$——对各种存储器、输入输出端口发出请求(Request)

(2) $\overline{RD}$、$\overline{WR}$——读、写数据的请求

(3) $\overline{BUSAK}$——确认 BUSRQ 信号

等等。这里，"—"意为低电平有效。本书中统一用 H 和 L 来表示电平在逻辑意义上的高和低。

2. CPU 与内部、外部设备的操作

微处理器与内部、外部设备之间的操作如图 2.2 所示。

(i) 开始时，用地址总线的高位经译码器选择将要存取的 IC 芯片(存储器、I/O 板等)，称为芯片使能$\overline{CE}$或片选$\overline{CS}$。

(ii) 把$\overline{IORQ}$(或$\overline{MREQ}$)置为 L，以此发出对 I/O(或内存)进行存取的请求。

(iii) 在此期间发出读$\overline{RD}$或写$\overline{WR}$请求。

(iv) 若为读指令，I/O 端口的并行数据通过数据总线被读入到 CPU，若为写指令，则并行数据通过数据总线从 CPU 送到 I/O 端口。

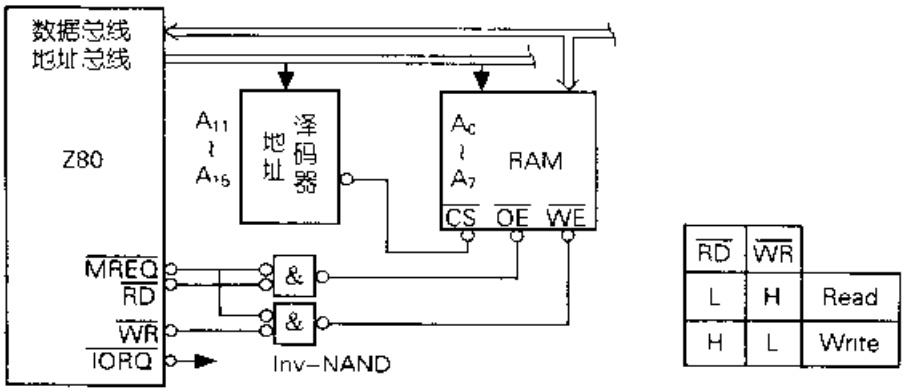
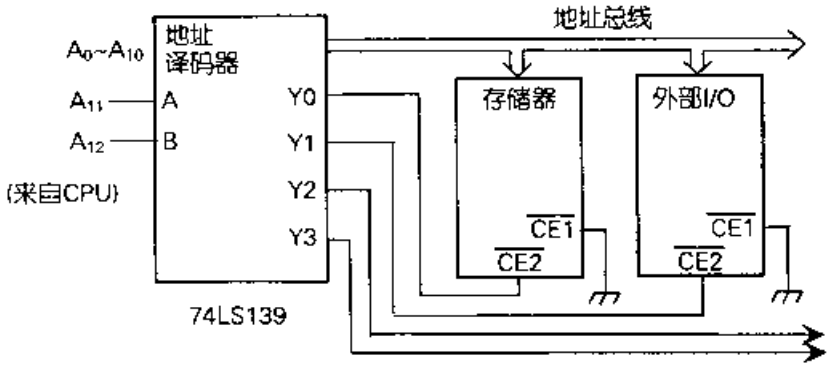


图 2.2 Z-80 MPU 和外部设备[9]

2.1.3 片 选

现在我们讨论一下图 2.2 所示的地址译码器的功能。从 CPU 引出的总线连接着大量的外部设备如存储器(RAM、ROM)和 I/O 芯片(如 8255A)等，CPU 要在必要时选择一个适当的 I/O 芯片进行操作(读出或写入)。使用图 2.3 所示的 SN74LS139 及 138 芯片，根据部分地址总线如 A_{11} 和 A_{12}，我们可以很容易设计出片选电路$\overline{CE}$[10]。



(a) 电路图

(输入端)			(输出端)			
G	B	A	Y0	Y1	Y2	Y3
GND	A_{12}	A_{11}	$\overline{CE2}$	$\overline{CE2}$	$\overline{CE2}$	$\overline{CE2}$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) 74LS139的功能表

图 2.3 地址译码器(74LS139)[10]

例题 2.1 利用图 2.3 所示的片选芯片 SN74LS139 选择外部 I/O 芯片时,地址是多少?

❖解答❖ 根据图示可知,外部 I/O 芯片的片选端$\overline{CE2}$与 Y1 相连,当 74LS139 的输出端口 Y1 变成 L 时,输入端口 B 为 0,A 为 1。因为 B 与地址线 A_{12}、A 与地址线 A_{11} 相连,来自 CPU 的 16 位地址数据应该是 0800h,h 表示 16 进制。

2.1.4 Z-80 PIO

PIO 是通用并行控制器(Parallel Input/Output interface),采用 8255 和 8253 等符号表示。

1. 8255

8255 是一种 PPI(Programmable Peripheral Interface)接口芯片。在第 4 章的用 C 语言实现接口控制部分有关于 8255 芯片的详细介绍。Z-80 PPI 有 3 个 8 位的输入输出端口,分别为 PA、PB 和 PC,利用 TTL 电平可以实现输入输出的双向传输。图 2.1 中的 PA 和 PB 等即是 PPI 的输入输出端口。

2. 8253

这是称为 PIT(Programmable Interval Timer)的通用定时器,内含 3 个 16 位的计数器,分别为计数器#0、#1 和#2。

2.1.5 握手信号

CPU 与外部设备的工作速度有很大差异。因此,数据的传送方和接收方必须能够一边确认对方的处理状态一边传送数据。这种传输方式称为**握手传输**。确认状态主要依赖以下 2 种信号:

(1) 选通脉冲信号——STB 是请求传送数据信号。

(2) 应答信号——ACK 是已收到数据、处理结束的应答信号。

Z-80 在执行打印任务时,CPU 先向串行打印机发出 STB 请求,然后再发送数据。打印机完成打印任务后则向计算机返回 ACK 应答信号。

2.2 微处理器软件

2.2.1 汇编语言

1. 微处理器的指令

微处理器的指令基本上是由 16 进制形式组成的机器语言。相比之下,汇编语言则与我们日常使用的自然语言更为接近。事实上,汇编语言是由操作指令的操作部分(运算部分)和标号部分(地址)经符号化而得到的语言,比机器语言更容易理解。当然,汇编语言程序最终需要翻译成机器语言程序,完成这种翻译工作的程序称为"**汇编程序**"。

图?.4反映了汇编语言源程序经过汇编程序的翻译,转换到目标程序,即机器语言程序的过程。这里的目标程序是指可执行程序[9]。

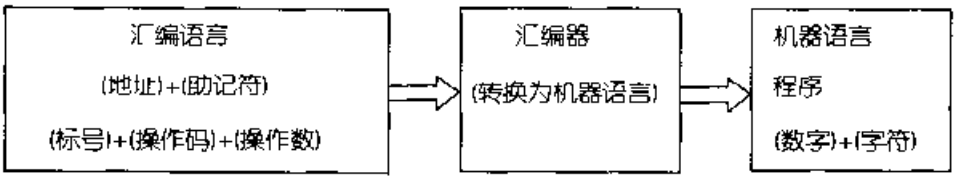


图 2.4 汇编语言

2. 基本指令

可以说,汇编语言基本是由"标号"及助记符组成的,基本要素是"操作码 操作数",具体含义如下:

(1) 标号与 FORTRAN 语言中的地址具有相同含义。

(2) 操作码是运算指令。

(3) 操作数由源操作数和目的操作数组成。

(4) 操作指令和数据是 8 位的。

例如,有如下指令:

操作码	目的操作数	源操作数
LD	A,	24h

这条指令的功能是"将 24h(16 进制的 24)送入寄存器 A"。在机器语言中,"LD A"的 16 进制代码是 3E,因此,完整的指令可用 16 进制表示为"3E24"。

在 Z-80 系统中,这样的基本指令(操作码)共有 64 个,所有操作数的形式都集中在一个表中。表 2.1 是基本指令的汇编语言和机器语言示例[10]。

表 2.1 Z-80 的指令集粹

助记符		机器指令	注 释
LD	A, n	3En	用 16 进制数 n 设置寄存器 A,n 为 2 位 16 进制数
LD	B, n	06n	用 16 进制数 n 设置寄存器 B(Load)
LD	B, A	47	寄存器 A 送入寄存器 B
LD	(mn), A	32 nm	寄存器 A 送入地址 nm
ADD	A, B	80	寄存器 B 累加到寄存器 A
ADD	A, C	81	寄存器 C 累加到寄存器 A
SUB	n	D 6n	从寄存器 A 中减去 n,n 为 8 位常数
INC	A	3C	寄存器 A 加 1
DEC	A	3D	寄存器 A 减 1
JP	mn	C3 nm	无条件跳转到地址 mn,mn 为 16 位的常数
JP	Z, lm	CAml	符合条件跳转到地址 lm
JP	PE, lm	EAml	大于或等于则跳转到地址 lm
IN	A, (n)	DBn	从地址 n 中指定的 I/O 端口输入
OUT	(n), A	D3n	将寄存器 A 输出到地址 n 指定的 I/O 端口,n 为端口的地址。

例题 2.2 用汇编语言程序实现运算 5 + 12。

❖解答❖

```
LD      A,      5h      ;设置寄存器 A 的值为 5h
LD      B,      Ch      ;设置寄存器 B 的值为 Ch( = 12)
```

```
ADD     A,      B       ;A+B的结果存入寄存器A
                        ;这种方式称为累加方式
```

2.2.2 汇编语言和机器语言的对比

1. 伪指令

```
ORG 800h ;表示从800h地址开始执行程序
END      ;程序结束
EQU      ;等值
```

2. 标 号

标号是标记一个地址的文字符号。注释则以";"做前缀标记。

3. 圆括号()的含义

操作数中的"02h"和"(02h)"具有不同的含义,"02h"直接表示该数值所指出的地址,而"(02h)"表示地址02h所指的内存单元中存储的内容。

4. 对PIO的输入输出

此处先对下述的输入输出端口操作程序做些解释。由于8255的各端口都含有控制寄存器和数据寄存器,因此,程序中事先做了标号定义。

(1) 分别用PAD和PAC定义端口PA的数据寄存器和控制寄存器。

(2) 分别用PBD和PBC定义端口PB的数据寄存器和控制寄存器。

(3) 将不同的地址分配给标号,此处为00h~03h。

(4) 在8255的工作模式中,4Fh表示8位全输入,0Fh表示8位全输出。通常,端口A全部作为输入而端口B全部作为输出。

```
                    ;(标号)   (操作码)   (操作数)        (注释)
                    PAD:      EQU       00h         ;PAD=00h等的定义
                    PAC:      EQU       01h
                    PBD:      EQU       02h
                    PBC:      EQU       03h
;(地址)  机器语言
                              ORG       600h        ;从相对地址600h开始执行
0600     3E4F       INIT:     LD        A, 4Fh      ;4Fh送入寄存器A,
                                                     ;设置端口A为全输入
0602     D301                 OUT       (PAC), A    ;寄存器A送入地址01中的
                                                     ;内容所指定的地址中
0604     3E0F                 LD        A, 0Fh      ;( )表示内容
0606     D303                 OUT       (PBC), A    ;初始化PBC为全输出
```

```
0608    DB00    MA00:   IN      A, (PAD)    ;将 0 地址的内容所确定的
                                            ;端口送入寄存器 A
060A    D302            OUT     (PBD), A    ;寄存器 A 输出到 PBD 的
                                            ;内容指定的地址
060C    C30806          JP      MA00        ;无条件跳转
                                            ;无限循环
060F                    END                 ;程序结束
```

例题 2.3 机器语言中的 0611 指令具有什么含义?

❖解答❖ 根据表 2.1,06 的含义为"LD B:加载寄存器 B",因此,该指令的功能是用 11h 加载寄存器 B。

2.3 Z-80 接口

2.3.1 练习用的 Z-80 的 I/O

前述的图 2.1 是作者将一个学习用过的 Z-80 微处理器的附属输入输出接口板的连接状态,做了简化处理之后的示例。事实上,该图中的 PA 和 PB 即是 8255 的端口 A 和 B。在 8255 上连接了许多的元器件,如 LED、继电器以及开关等。在部分端口上连接着 1 个 AD 转换器,来自传感器的信号经 AD 转换器转换后变成 0、1 信号,再输入到 8255(只使用了 4 位)。此板上还有另一个 8255,开放给用户,供其进行电路测试。借助已学习过的地址译码器,可以选择其中的任何一个 8255。

1. LED 的发光电路

图 2.5(a)所示为 LED 的发光电路。电路中接有 $+V_{CC}$(最大不超过约 12V 的直流电源),因为 H(高电平)有效,故称之为上拉电路。在电子电路中,数字 IC 的输入输出总是以电平的 H 和 L 状态保持着。8255 的输出是 H 时,反相器的输出变成 L,则 LED 发光。电阻 R 需要根据 LED 中通过的电流(15～20mA)来计算,通常在 200～300Ω 的范围。

2. 触发开关电路

图 2.5(b)所示为触发开关电路的示意图,通常也使用这种上拉电路。

(1) 当开关断开时,7404 的输入为 H 而输出为 L。

(2) 当开关闭合时,接点为 L,经反相器的反转变成 H 后送入 8255。因为在触发开关中几乎没有电流通过,故上拉电阻的值大约应为 10k～20kΩ。由于机械开关切换时,会出现振荡,产生数字噪声,故应该加入防止振荡的电路。

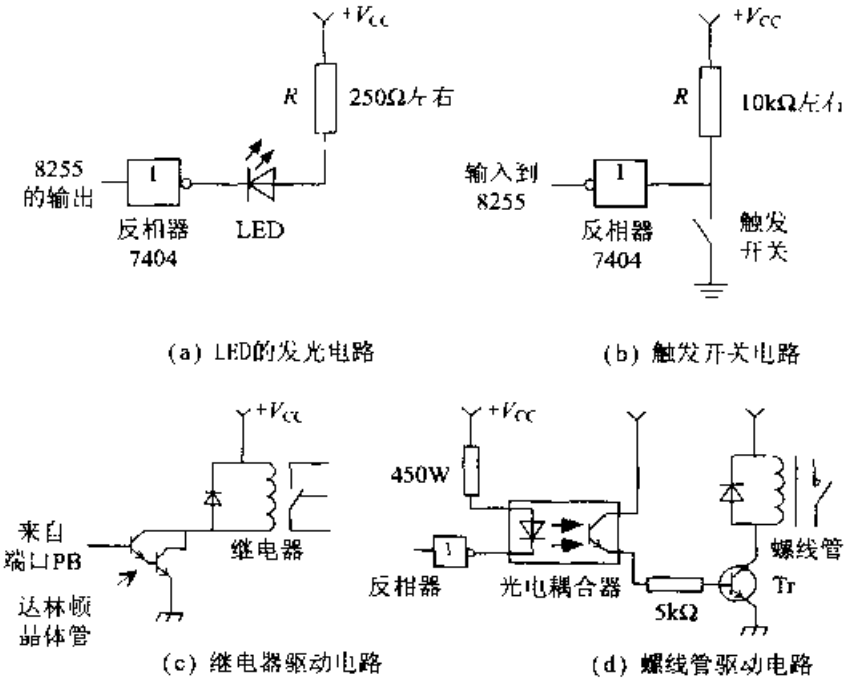


图 2.5 各种接口[21]

3. 继电器驱动电路

图 2.5(c)所示为继电器驱动电路示意图。其中,螺线管 μPA2003 在 IC 内部置入二极管,可以吸收继电器断续时引起的电涌电流。驱动电流可以限定在数毫安以内。由于 PPI 芯片不能直接驱动电子器件,所以需要通过此类的晶体管和 OP 放大器(运算放大器)进行缓冲。

4. 螺线管驱动电路

图 2.5(d)所示为利用光电耦合器将微处理器和螺线管隔离,以防止电气噪声的示例。光电耦合导通时,晶体管 Tr 也被导通,使电流在螺线管中通过。光电耦合隔离器包括光电晶体管耦合器和光电晶闸管耦合器。

5. 扬声器的驱动

在图 2.1 所示的扬声器电路中,通常要采用 OP 放大器进行电流放大,才能驱动扬声器工作。

练习 2

问题 2.1 在下面的[ ]内填空。

(1) 以 8 位为单位进行数据处理的微处理器很多，将 8 位组成 1 个存储单元称为[ ]。

(2) 用 8 位组成的无符号数据的最大值是[ ]。

(3) 用 8 位组成的有符号数据的最大值是[ ]，最小值是[ ]。

(4) 输入的数据 A 和 B 都是 1 时，使输出为 0 的电子元件是[ ]。

(5) 输入数据 A 和 B 具有不同的值时，使输出为 H 的电子元件是[ ]，将其写成逻辑表达式是[ ]。

问题 2.2 假定使用 8 位地址总线的 A_0、A_1 和 A_2，若构成的值为 5h，则将开关 #0 的状态传送到 CPU。试设计满足此要求的地址译码器。

问题 2.3 将下面的汇编代码翻译成机器指令。

(1) LD B, 13h　　(2) SUB 21

问题 2.4 解释下述程序，并翻译成机器语言代码(解释部分填在[ ]内)。

```
;(地址) (机器指令)

[(2)]  [(3)]
1002   [(5)]
1004   [(6)]
1006   D603
1008
```

```
;(标号)  (操作码)  (操作数)  (注释)
PB:      EQU       03h
         ORG       1000h     ;从[(1)]开始执行
         LD        A,10h     ;设置[(4)]
         LD        B, 01h
         ADD       A, B      ;[(7)]
         SUB       PB        ;[(8)]
         END                 ;答案是
                             ;[(9)]
```

问题 2.5 使用课文中所述的输入输出端口。将端口 A 的开关模式传送给端口 B 的 LED，试编写控制 LED 循环左移发光的程序。假定端口的地址为 04h～07h。

问题 2.6 译码器 IC74LS139 的真值表如图 2.3 所示。将地址线(A_6、A_7)连接到 74LS139 的(A、B)上。试设计一个电路，使得在 $\overline{IORQ}$ 为 L 时，若地址线(A_6、A_7)为(1、0)可以选择 AD 转换器，若为(0、1)则可以选择 DA 转换器。

问题 2.7 编写一个汇编语言程序计算$(21+7)_{10}$，程序从 1000h 地址开始执行。然后，再将其翻译成机器语言指令。

chapter 3

检测控制领域使用的C语言

众所周知，计算机是通过软件来操纵的。在现今的Windows时代，随着应用软件的不断进步，迅速实现常规作业和通信已成为可能。不过，当您从事某些科学计算和机器人控制等工作时就会发现，能够直接购买到适用的商品软件的情况实在是很少见的，这促使我们必须用C、C++或汇编等计算机语言来编制自己的程序。本章中我们将有针对性地学习一些C语言的有关知识，可以说，这些内容是用C语言开发计算机检测和控制接口程序的精华部分。至于语言的版本，可以考虑使用MS-VC（Microsoft Visual C++）等C++语言。除了这些语言知识之外，本章中还给出了一个记数板（编码板）的控制程序实例。

3.1 概 述

3.1.1 为什么使用C语言

目前,C语言的使用已相当普遍,检测、控制、图像处理和通信乃至游戏等几乎所有领域都采用C语言进行程序设计。最初,C语言只是为了对UNIX操作系统进行描述而开发的,现在则早已移植到了MS-DOS(或者说DOS/V)系统中,并构成了所谓BIOS-DOS-C/C++开发平台,这使语言本身也变得日益重要。此外,C语言不仅可以在微机或者大型计算机和超大型计算机上使用,甚至也被移植到了便携计算机上。

C语言的普及主要基于如下原因:

(1) C语言自身的基本命令较少,是一种很"小"的语言。

(2) 结构简单,能够很好地适应科学计算及机械系统控制领域的要求。

(3) 对于同一个任务,随着对语言掌握的熟练程度不同,可以给出多种不同的描述方法,极富弹性。

(4) 处理速度快,仅次于汇编语言,比BASIC语言快近10倍。

(5) 与汇编语言不同的是,C语言是高级语言,而汇编语言是低级语言。

在C语言的学习中,指针始终是需要重点掌握的内容。这里我们将介绍一些控制领域中使用频繁、程度也稍高些的编程内容。从能力上说,可以认为C语言是介于汇编语言与高级语言之间的"中级语言"。在使用C语言进行软件开发时,可以先从小的程序块作起,调试并确认其正确性,再逐步积累到较大的程序,这不失为一种行之有效的方法。

3.1.2 字符串

虽然"abc"是由3个字符组成的字符串,但在存储时必须定义长度为4的char类型数组,即:

```
char  moji[ ] = {"abc"};
```

在内存中,系统按{'a','b','c','\0'}的顺序存储这些字符,且为了与其它单元分隔,系统自动添加了字符'\0'作为字符串的结束符。因此,moji[4]是一个含有4个字符的数组,最后的字符'\0'是系统附加上的结束标志。

1. 关于'\0'和'0'

(1) 字符"'\0'"用在字符串的最后表示字符串结束,与数值 0 及 0x0 含义相同。

(2) 字符"'0'"是 ANSI 字符(即 ASCII 码字符),数值是 0x30。

3.1.3 1 维空间分配

考虑如下语句组:

```
int *   p;
p = (int *)calloc(500, sizeof(int));
```

此语句组的作用是向系统申请"整数型字节数(2B,2 个字节)×500 的内存空间"。

再考虑如下语句组:

```
char *   p;
p = (char *)malloc(1000);
```

此语句组申请 1000 字节的 char 类型的内存空间。当 malloc 函数分配内存成功时,内存块的首地址被存入指针变量 p;若分配失败,p 的值是函数的返回标志值 NULL。

sizeof 是以字节为单位测试一个数据或数据类型所占用的存储空间大小的运算符。可以参考如下代码使用 sizeof 运算符:

```
char   c;
int   s, a, b[5][10];
s = sizeof c;              /* s=1B          */
s = sizeof a;              /* s=2           */
s = sizeof b;              /* s=2×50×10     */
s = sizeof b[0];           /* s=2×10        */
s = sizeof(long);          /* s=4           */
```

3.1.4 指 针

指针变量是只能以地址为值(即存储地址)的变量。使用时应注意如下的一些细节:

(1) & 是取变量地址运算符。

(2) * 是引用变量值运算符,即引用一个地址所指向的内存单元中存储的值。

(3) 数组名是指针。

(4) 指针变量具有与普通变量相同的使用方法。

表 3.1 体现了指针和普通数据的关系。

表 3.1 指 针

定 义	数据引用	指针引用
int a	a	&a
int * p	*p	p

1. 指针应用的简单例子

以下是通过指针引用变量的代码:

```
int * a, b, c;
a = &b; b = 3; c = *a;                    /* c=3 */
```

2. 数组和指针

(1) 下述代码体现了指针与数组的关系:

```
int a[20]; char *p;
p = a; p = a+1; p = &a[3];                /* 都是正确的语句 */
```

(2) 字符串与指针

使用指针操作字符串是很方便的:

```
char ss[ ] = "abc";
char *p ="abc";                           /* 二者相同 */
```

3. 2维数组和指针

若定义如下的数组:

```
char ss[3][10]
      = {"Sunday", "Monday", "Tuesday"};
char *p[3];
      = {"Sunday", "Monday", "Tuesday"};  /* 二者相同 */
```

则有

ss[0]和 p[0]的内容是 Sunday\0,

ss[1]和 p[1]的内容是 Monday\0。

例题 3.1 使用指针输出数组 c1[ ]的第 3 个及以后的所有元素。

❖解答❖

```
#include <stdio.h>
```

```
char c1[ ] = {"abcdefgh"};
void main( )
{
        char * p;
        p = &(c1[2]);        /* 指定数组的第 3 个元素的地址 */
        printf("\n%s", p);   /* 输出结果为 cdefgh */
}
```

4. 函数和指针

在 C 语言的程序中,调用函数与被调用函数可以有很多种交换数据的方法,指针则是最常用的技术。应该注意到数组名是指针。下图中的左侧是调用函数,右侧是被调用函数。

void main( ) /* 传递数据的地址 */
{
  int a, b;
  function(&a,&b);
}
	void function(int * c, int * d)
	{
	  int x;
	  c + = 1; d/=2; /* 数值变化 */
	}

void main( ) /* 传递数组 */
{
  char a[4];
 	
  function(a);
}
	void function(char b[ ])
	{
	  b[0] ='b';
	  b[1] ='c'; /* 用其它字符赋值 */
	  printf(…);
	}

void main( ) /* 传递 2 维数组 */
{
  int aa[3][10] = {…;
  function(aa);
}
	void function(int bb[ ][10])
	{
	
	}

void main( ) /* 传递字符串 */
{
  function("xyz…");
}
	void function(char * po)
	{
	  po[1] ='a'; /* 替换一个字符 */
	}

例题 3.2 利用传递并接收指针变量的方法,在被调用函数内计算 c = a + b,并将结果返回给调用函数(此处为主函数)。

❖解答❖

```
#include <stdio.h>
void fun0(int *x, int *y, int *z);          /* 函数定义 */
void main( )
{
  int a=2, b=5, c;                          /* 变量初始化 */
  fun0(&a, &b, &c);                         /* 使用指针调用函数 */
  printf("\n%d %d %d", a, b, c);            /* 显示结果为 2 5 7 */
}
void fun0(int *x, int *y, int *z)           /* 函数定义 */
{
  *z = *x + *y;
}
```

5. 矩阵和2维指针

在控制领域中需要经常使用矩阵运算。虽然可以采用类似FORTRAN的风格设计矩阵运算程序,但在C语言中可以使用指针使其运算速度更快。这里假定方阵a的阶数m满足$m<$Max。初始的数组元素输入时,可以用空格或回车作数据之间的分隔符。下述的代码用于将矩阵a的元素复制给矩阵b,其中的malloc用于动态分配内存,程序中较为重要的内容加了虚线框。

```
/* mat move.c: use 2   dimensional pointers */
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <malloc.h>          /* malloc 的头文件 */
# define Max 10
double a [Max][Max], b[Max][Max], c[Max] [Max] ;
int  , m;
void mat_printf (char ch[20] , double  **A, int m, int n )
                              /* 显示矩阵函数 */
                              /*  ** A 是 2 维指针 */
{
      int i, j;
      printf ("%s\n", ch);
      for(i=0; i<m ; i++)
      {  for (j = 0 ; j < n ; j ++)
           printf ("%lf ", A[i][j]);
      }
}
```

```
/* Matrix move : from A to B */
void mat_move(double * * A, double * * B, int m, int n)  /* A、B为2维指针 */
{
      int i, j ;
      for(i=0 ; i<m; i++ )
      {  for (j=0 ; j<n; j++)
            B[i][j] = A[i][j];
      }
}
void main (void)                          /* 主函数 */
{  int i, j, k, errx ; char cc[20];
   double * *pa, * *pb;                   /* 2维指针 */
      printf ("/nelements m= /?");  /* 矩阵阶数 */
      scanf ("%d", &m);
      gets(cc) ;                          /* 用gets( )清字符缓冲区 */
      l= m;
      printf ("\nread matrix a/n") ;  /* 矩阵元素 */
      for(i=0; i<m ; i++)
      {  for (j=0; j<l; j++)
            scanf("%lf", &a[i][j]) ;
      }
pa = (double  * *)malloc(Max *Max * sizeof(double *)) ;
                                          /* 2维空间分配 */
pb= (double * *)malloc(Max *Max * sizeof (double *));
   for(i = 0; i<m; i++)
   {  pa[i] = a[i] ; pb[i] =b[i] ; } /* pa[i]是a[i] [0]的指针 */
   mat_move(pa, pb, m, l) ;
   printf("\n") ;
   mat_printf("a", pa, m, l) ;
   mat_printf("b", pb, m, l) ;
}
```

程序中的函数 mat_move( )可以原样使用到别处。

3.1.5 关于 scanf

使用 scanf 函数时需要注意以下问题：

(1) 输入数据中的空格符被舍弃(即不能输入含有空格字符的字符串)；如果需要原样读入输入的一行字符可使用 gets 函数。

(2) 读入数据后输入缓冲区中会残留一个多余的'\n'字符：

可以用 gets 将其读入并舍弃。

(3) 遇到不能转换的字符时终止读入。

可使用 gets 代替 scanf 读取数据。

为了防止接收数据出错,可以使用如下方法:

(4) 对于数值型的数据,可以先以字符串方式读入,然后再进行数据转换:

```
gets(buf);
d=atoi (buf); /*  其它类型的数据转换函数还有 atol、atof、itoa 和
                  gcvt 等 */
```

(5) 用 sscanf 函数从缓冲区而非键盘读入数据:

```
gets(buf);
sscanf(buf, "%d", &d);
```

3.2 检测控制中常用的C语言知识

3.2.1 文件管理

在控制过程中,常常会产生大量的 AD 转换数据和中间数据,需要以二进制方式对磁盘进行存取。二进制数据是指定点数和浮点数的二进制形式,当数据量较大时采用二进制方式可以得到更高的处理速度。少量的初始化数据也可以按字符方式读入,此时通常使用标准 ANSI 字符序列,这些字符可以直接从控制台输入并有简单的编辑能力。

1. 字符数据的读写

假设在磁盘上已创建了一个包含初始值的文本文件 da1.tab,其结构和内容如下:

```
Example   1
123                  //数值 1
234                  //数值 2
```

以下是从 da1.tab 中只将数值数据读出并写入文件 da2.tab 的程序:

```
# include <stdio.h>
# include <stdlib.h>
char file1[40]= {"da1.tab"};          /*  Table name */
char file2[40] = {"da2.tab"};
void main( )
{
    char buf[80]; int d;
```

```
FILE * fp10;                          /* 打开读文件 */
if((fp10 = fopen(file1, "r")) ==NULL)
{
    printf ("\nopen-err!") ;
    exit (1);
}
fgets(buf, 80, fp10) ;                /* 读入第一行字符串"Example 1"并舍弃 */
fgets(buf, 80, fp10) ;                /* 读入第2行 */
d = atoi(buf);                        /* 将字符串"123"转换成整数 */
fclose(fp10);                         /* 关闭文件 */
   fp10 = fopen(file2, "w");          /* 打开写文件 */
   fputs(buf, fp10) ;                 /* 写入字符串和 CRLF */
   fprintf(fp10, "%4d", d) ;          /* 写入数据 */
   fputs("\n", fp10) ;                /* 写入 CRLF */
   fclose(fp10);                      /* 关闭文件 */
}
```

其中,文件打开时的方式字符的含义为:

"r":文本方式只读(fscanf、fgets用)

"w":文本方式只写(fprintf、fputs用)

"a":文本方式追加

2. 二进制数据的读写

```
#include <stdio.h>
#include <stdlib.h>
char name1[ ]= {"read_file"} ;                    /* 初始值设置 */
char name2[ ] = {"write_file"};
void main( )
{   float data[512] ;                             /* 1个记录为512 x 4字节 */
 FILE *fp33 ;
 if ((fp33 = fopen(name1, "rb")) ==NULL)          /* 以二进制方式打开文件读 */
 {  printf("\nread err!") ;
    exit(1);}
 fread (data, 512*4, 1, fp33) ;                   /* 读出第一个记录 */
 fclose(fp33) ;
if ((fp33 = fopen(name2, "wb")) ==NULL)           /* 以二进制方式打开文件写 */
{  printf("\nwrite err!") ;
   exit(1) ;
}
fwrite(data, 512*4, 1, fp33) ;                    /* 写入第一个记录 */
fclose(fp33) ;
}
```

其中,文件打开时的方式字符的含义为:

"rb":二进制方式只读(fread用)
"wb":二进制方式只写(fwrite用)

3.2.2 字符串操作

编程中,时常需要将一个符号转换为dat和tab文件名(即附加上dat或tab作为文件的扩展名)。以下的示例代码可以在输入a1或a1.tab等字符串时,自动生成a1.tab作为输出结果。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char buf[40];
char tab[ ]= {".tab"} ;
void main( )
{
    int i, n ;
    scanf("%s", buf);
    n=strlen(buf) ;                                  /* 测试字符串的长度 */
    for(i=0; i<n; i++)
    {
        if(buf[i] ==0x2e ||buf[i] ==0x20)            /* "."字符或者" "(空格)字符 */
        {
            buff[i] = '\0';
            break;
        }
    }
    strcat(buf, tab);                               /* 将字符串".tab"连接到字符串末尾 */
    printf("\n%s", buf);
}
```

3.2.3 位操作

作为示例,从LSB(低字节)算起,将第3位置1和清0。

```
#include <stdio.h>
#include <stdlib.h>
unsigned int n0, n1;
void main( )
```

```
{
    int i, n;
    n0 = n1 = 0xf0;                 /* 11110000 */
    n0 = n0 | 0x4;                  /* 第3位置1 */
    n1 = n1 & 0xfb;                 /* 第3位清0 */
    printf("\n%x %x", n0, n1);      /* 结果为n0=0xf4,n1=0xf0 */
}
```

3.2.4 键盘输入

在检测控制中,经常需要通过键盘输入控制程序的启动、分支和结束等过程。下述程序中的头文件 discon.h 是 DOS/V 系统使用的显示控制例程,列于附录 B。代码中主要体现的问题包括:

(i) 启动:用 gets( )等函数等待输入

(ii) 程序的暂停和继续

```
/* wait until key_in;有键盘输入时程序暂停 */
#include <stdio.h>
#include "discon.h"             /* 参见附录B */
void main( )
{
    push_key(3);                /* 用天蓝色显示,等待输入 */
    }
    void push_key(int color)
    {
    ccolor(color);              /* discon.h中的函数 */
    locate(1, 24);              /* discon.h中的函数 */
    printf("* * * 请按任意键继续 * * *");
        getch( );               /* 等待按键 */
    printf("\n");               /* 参照discon.h中的函数 */
    clrscr( );
    cur_on( );
    ccolor(7);
}
```

(iii) 利用键盘输入终止循环

```
while( !kbhit( ))
{                       /* 如果有键按下,结束{   }内的处理 */
    处理;               /* kbhit( )是C语言的库函数 */
}                       /* 按键则结束循环 */
```

(iv) 判别是否曾有键按下

在上面的例程中,循环中一直等待输入使其结束的按键,一旦有键盘输入,直接中止循环过程。事实上,很多时候循环中可能还有某些计算,尽管有键按下,程序也只能在适当的时候(如计算结束)才能处理此按键,并判定是否继续进行循环。为此,可以采用下例中的方法(DOS/V机及PC-9800系列用)。

```
union REGS inregs, outregs;          /* C的寄存器共用体定义 */
struct SREGS segregs;                /* C的寄存器结构体定义 */
void main( )
{
    ……                               /* 即使有按键也继续处理 */
    if(keyb_status( ) != 0)          /* 若有键盘输入则进行处理 */
        goto h000;                   /* 此时才进行判定 */
    ……
h000: ……                            /* 循环内可进行分支处理 */
}

/* 用键盘BIOS判别键的状态:调用时执行软件中断 */
/* 返回值:255(data exist in buffer),
      0(data no-exist in buffer) */
int keyb_status(void)                /* 按键的判定函数 */
{
    inregs.h.ah = 0x0b;              /* 设置ah寄存器 */
    intdos(&inregs, &outregs);       /* BIOS软中断调用 */
    return (outregs.h.al);           /* 系统会设置al,函数返回此值 */
}                                    /* 若outregs.h.al有值,说明有键按下 */
```

例题 3.3 给出通过按键开始执行,再次按键结束运行的示例代码。

❖解答❖

```
gets(buf);                  /* 等待按键 */
while(!kbhit( ))            /* 若有键按下,结束!{ …… }循环 */
{ …… }
```

3.3 知识扩展

3.3.1 结构体

一个人的姓名、年龄和身高等，可以分别用 char、int 和 float 类型的数据描述，但有时需要将这些数据合在一起作为"人"的资料整体进行文件管理。或者，材料的长、宽、高等尺寸信息，机器人的各关节点的转角、角速度、力和转矩等信息，若将它们各自作为一个整体看待将更有助于理解。为此，可以使用结构体，定义形式如下：

(1) struct 结构体名
{ 成员；
…；
}； /* 结构体变量随后定义 */

或者将结构体类型和变量一起定义：

(2) struct 结构体名
{ 成员；
…；
} 变量名； /* 结构体变量同时定义 */

等等。

(3) 如果需要，也可以利用下述程序所示的方法，定义结构体类型的指针。通过结构体指针引用成员时使用"->"运算符。

C 语言的结构体是 C++ 语言中 class(即类)的原型，可想而知，以后使用结构体的机会是很多的。下述程序是使用结构体及其指针的一个示例。

```
#include <string.h>
struct A000 {  char a0[20] ;          /* 结构体定义 */
               int a1;
               double a2 ;
            };
      void main( )
      {
          struct A000 a000 = {"神奈川一", 50, 165.4 };
                                  /* 利用前面定义的结构体类型定义变量 */
              strcpy(a000.a0,"东京二郎");                    /* "."运算 */
```

```
        a000.a1 = 51;
    struct A000 *c00 = &a000;                    /* 设置指针:使用->运算符 */
        (*c00).a1= 55;                           /* 使用"."运算符的场合 */
        c00->a1= 55;                             /* 使用"->"运算符的场合*/
    struct A000 b000[5] ={{"大阪", 0, 1 0}, {…}, …};
                                                 /* 结构相同的5个元素 */
        c00= b000;
        c00->a2= 175.6;    /* 因为c00是指针,所以使用"->"运算符 */
        function (struct a000);                  /* 结构体做函数参数 */
}
void function(struct A000 x00)                   /* 形式参数是结构体类型 */
{
        x00.a1 = 100;
}
```

3.3.2 内嵌汇编

在C语言程序中,经过说明可以直接插入汇编语言程序段。若使用MASM(Miscrosoft Assembler)需要以_asm { }进行说明,而使用TASM(Turbo Assembler)时可以直接使用asm作为修饰符,不需要下划线。为了实现对硬件的控制,或提高速度及节约内存,常常需要使用汇编语言代码。

1. MASM的形式

在C语言程序的函数体内,可以按如下形式插入汇编语言程序段:

```
_asm                          /* 内嵌汇编说明 */
{
    mov ah, 2                 /* 将2送入ah(8位寄存器) */
    mov al, 7                 /* 将7送入al(8位寄存器) */
    Int 21h                   /* 软件中断调用 */
}                             /* asm结束 */
```

2. 扩展寄存器

PCI是32位(或32位以上)总线,但C语言编译器中的寄存器全部为16位,没有定义32位寄存器。这是件非常麻烦的事。为了在汇编器中(80386以上机型)追加上32位的寄存器ecx和edx,可以使用如下的行内(内嵌)汇编代码:

```
unsigned long int la;
_asm
{
    mov ecx, la           ;传送32位值
}
```

编译含有这样代码的程序时，必须使用>tcc - 3 - B xx.c 形式的编译开关，其中：

(1) 编译开关-3 是 80386CPU 的保护模式转换命令

(2) 编译开关 - B 是表示调用外部汇编器处理内嵌汇编语句

3.3.3 内存管理[13]

C 语言可以采用从 small 模式到 huge 模式的编译模式，但由于每个寄存器都是 16 位的，故直接管理的代码和数据都不能超过 64KB。在使用 small 模式编译时，程序的长度不能超过 64KB。若程序较大，超过了 64KB，可以采用下述方法处理：

(1) 将程序细分为小的模块，每块单独编译成 obj 文件，再将这些 obj 文件连接成可执行文件。或者：

(2) 采用 large 模式或 huge 模式进行编译(使用 large 模式时数组的大小不能超过 64KB，但 huge 模式对此没有限制)。

以下是一些定义的示例：

```
char _far a;
char _huge *b;
char _far func( );          /* func( )是一个返回字符的函数，具有 32 位地址 */
char (_far *pp)( );         /* pp 是一个 far(32 位)指针，用于指向返回字符的函数 */
```


3.4 编译和编辑环境

编译(compile)是将与我们日常书面语言接近的计算机语言，翻译成计算机可以识别的机器语言，并得到可执行程序的过程。在自己购买计算机之后，通常可参考计算机语言方面的书籍进行程序设计，但究竟安装什么样的翻译软件(编译软件)，或安装后如何才能正确地操作等问题，尽管用户非常关心，但一般 C 语言的书籍中却很少讲解。若只靠实践去摸索，事倍功半。

本节中，我们将就如何处理 C 语言程序的方法和过程，对 Visual C/C++ Studio 和 MS-DOS 平台的操作环境做些讨论。

3.4.1 MS-VC++环境

应该说，虽然这里使用了 Windows 环境，并采用 Visual C++ Studio 进

行程序的设计和编译，但并没有使用该语言本身的MFC(Microsoft Foundation Class，基础类库)功能，而是采用了DOS平台的Win32编程方式(称为控制台应用程序)进行设计，以使其能够运行于DOS窗口或MS-DOS环境[14]。

(i) 在将Visual C++ ver.6.0等安装后，将Visual Studio.exe的图标显示在桌面上。双击该图标打开Visual Studio。

(ii) 在Visual Studio画面中，选择"File(F)"菜单中的"New(N)"菜单项。缺省的平台是Win32。

(iii)在New窗口中选择"File"页，单击"C++ Source File"选项，然后选择"Project"页，单击"Win32 Application"或者"ATL COM AppWizard"。

(iii')若要生成单独的C++源程序文件，可在New窗口中点击"File"页，再选择"C++ Source File"项，单击OK按钮，系统自动生成缺省的CPP1.cpp文件。此时，再选择主菜单的"File(F)/Save As(A)"菜单项，输入一个用户文件名即可将其存盘，得到的程序为xxx.cpp(以.cpp作为扩展名)形式。

(iv) 在Visual Studio主菜单的Tools(T)/Options(O)中选择"Directories(S)"页，并在弹出的Directories设置窗口中：

1) 对Include命令中涉及的文件，主要是必须将工程中的用户定义头文件(如xxx.h)的路径追加到此系统路径描述中。

2) 对库文件，即Lib文件，必须将工程中使用到的xxx.lib文件的路径追加到此系统路径描述中。

(v) 单击OK按钮，系统自动生成一个缺省的CPP1.cpp文件。选择主菜单的File(F)/Save As(A)菜单项，填入用户自己确定的文件名xxx，就可以将文件按文件名xxx.cpp(.cpp是C++源程序文件的扩展名)将文件存盘。

每个Project文件(工程文件)代表一个具体的项目，应该将用户自己新生成的属于该项目的文件(包括源程序文件、头文件以及资源文件等：译者注)添加到此项目文件中。

(vi) 若觉得上述文件操作不合适，选择主菜单的"File(F)"菜单中的"Close WorkSpace(K)"菜单项，即可以关闭当前的工程，重新开始。

(vii) 输入程序代码。

(viii) 选择"Build(B)"菜单项编译程序，再选择"Run(X)"菜单项运行程序。不过，应注意此方法不能处理随后5.1节中所述的包含软、硬件中断的程序[1)]。

1) 依译者所见，原文的这一部分内容不太清楚，事实上，在选择"New"菜单项时，VC弹出一个多页窗口，各页之间的选项都代表着不同的程序或文档类型，不会出现步骤(iii)中所述的连带关系。在选择其它页时，原来页中的选择也就自然的无效了。比较合理的作法是：在New窗口中选择"Project"页；单击"Win32 Console Application"选项；在窗口右侧的两个编辑域中输入项目文件名和选择一个存放将要生成的项目的文件夹；单击OK按钮，系统弹出一个新窗口；在新窗口中选择"A Simple application"(也可以是其它选项)；单击Finish按钮，系统又会弹出一个新窗口；单击OK按钮。经过上述步骤后，系统按用户指定的名字生成一个项目，并有一个main函数，用户可在其中添加程序代码。 —译者注

3.4.2 DOS 窗口环境

PC-9800 系列中可以分为 Windows 和 MS-DOS 两个 OS 驱动器，能够从其中任何一个驱动器启动系统，通常启动后的驱动器为“A:\系统目录”形式。在 DOS/V 机中，Windows 通常是以“C:\系统目录”的形式占据 C 盘的一部分。若在 Windows 系统中使用 MS-DOS 环境，可以参考下述的方法。但应该说明，根据具体情况不同，可能需要在 C 盘的根目录下的 config.sys 文件和 autoexec.bat 文件中添加第 7 章中所述的必要的 MS-DOS 命令(有时 Windows 可能死锁)。

1. MS-DOS 提示符

因为各种应用可以在 Windows 下共存，故可以将 DOS 窗口作为 Windows 的一个任务打开，且在任何时候都可以方便地切换回 Windows。但由于大量的 Windows 程序驻留内存，致使 DOS 能够使用的内存变得很少。此外，类似 DOS 中断处理之类的复杂程序在这种情况下是不能运行的。

可以考虑按下述方法将 DOS 图标移到桌面上以简化每次的启动操作：

(i) 启动 Windows，将 Explorer 图标移动到桌面上。具体的步骤是：用鼠标左键双击桌面上“我的电脑”图标；选择 C 驱动器后再双击 C:\Windows 文件夹；将鼠标指针移到 Explorer.exe 图标，在按下鼠标左键的同时并移动鼠标，就可以将此图标移动到桌面的空白处。

(ii) 同样地，在 C:\Windows 文件夹中还可以看到 MS-DOS 提示符，将该图标也移到桌面上。具体的步骤是：用鼠标双击 Explorer 图标；打开 C:\Windows 文件夹；用鼠标指针指向 DOS 提示符，按下鼠标左键同时将其移到桌面的空白处。

(iii) 双击 DOS 提示符图标，Windows 会在桌面上打开 MS-DOS 窗口。

2. MS-DOS 模式

此模式是指在 Windows 环境下全屏展开的 MS-DOS 画面。与以往不同，此时只有键盘输入有效，鼠标无效。内存中由于有 win.com(此程序负责结束 DOS 模式并运行 config.wos 和 autoexec.wos)等返回 Windows 环境所必须的程序驻留，导致留给用户程序使用的空间减少。从 Windows 切换到 MS-DOS 模式及终止 MS-DOS 模式的操作如下：

(i) 在 Windows 环境下，双击上述 1.中所代表的 MS-DOS 模式的图标。

或者，

(i') 从任务栏的“开始”菜单中选择“关闭系统”，再选择“重新启动计算机并切换到 MS-DOS 方式”。

在这种方式下不应执行含有 DOS 中断处理过程的程序，可以在 DOS 提示符下执行 exit 命令返回 Windows。

3. 用 F8 等功能键启动的 MS-DOS 窗口

(i) 由于在 Windows 的启动中要先运行 MS-DOS,因此,可以在启动期间一直按下 F8 键不放,也有的机器需要一直按下 Ctrl 键。不过,应注意误操作有可能导致 Windows 出错,所以,最好先向卖方询问清楚。

(ii) 启动菜单

系统接收到 F8 键后会显示如下的启动菜单:

Miscrosoft Windows xx Startup Menu[15]

① Normal———一般方式,根据 Winboot.ini 启动 Windows。

② Logged——参照日志中的记录启动 Windows。

③ Safe mode——此为安全模式。当键盘操作、config 设置等出现错误时,可以通过安全模式启动 Windows 进行错误修复。

④ Safe mode with network support——以安全模式启动并支持网络功能。

⑤ Step-by-step configuration——以单步执行配置命令方式启动。

⑥ Command prompt only——几乎以完全的 MS-DOS 模式启动,Windows 以最小方式驻留。在系统以 MS-DOS 方式正常工作后,可以再用"win ↓"命令启动 Windows。因为按此方式启动还不是完全的 MS-DOS,故称其为**DOS 窗口**。

⑦ Safe mode command prompt only——按 MS-DOS 模式启动系统,但不执行 config 文件和 autoexec 文件。

选择"⑥"虽然可以启动 DOS 窗口,但仍然不是完全的 MS-DOS,只是 Windows 以最小方式常驻而已。在这种情况下,仍不能执行使用了中断的 MS-DOS 用的 AD(模/数转换)板和视频显示卡等程序。同样,执行 exit 命令可返回 Windows。

4. 系统死锁时

系统死锁时,可以同时按下 Ctrl + Alt + Del 键重新启动计算机(热启动)。重启动时 Windows 可能正常运行,也可能出现前述的(3)中的启动菜单,此时可选择"③"即 Safe mode 启动,再将 config 文件修改到以前的正确状态。若机器不能热启动,可以按 Reset 键或切断电源并重新启动。

3.4.3 MS-DOS 窗口下的 C 程序编译

1. Turbo C++ for DOS(ver. 4.2 J)

(i) 进入 DOS 环境(包括 DOS 模式、DOS 提示符或完全的 MS-DOS,以下同)。

(ii) 安装 C 语言系统后,在根目录下的 Autoexec.bat 中追加以下内容,其中 path 是外部命令文件的查找路径命令:

```
path %path%;c:\TC4\bin          ;TC4 是 Turbo C++ 的目录
                                ;%path%表示原来的路径设置
```

(ii) 编译

```
>tcc xxx.cpp 及>tc -ml xxx.cpp  ; -ml 表示 large 模式
>tcc-c-ml yyy.c                 ; -c 表示只形成 obj 文件
>tcc-ml xxx.cpp yyy.obj zzz.lib ;obj 和库连接
>tcc-ml xxx.c @mylib            ; mylib 是由下述文件连接而成的
```

此处的 mylib 是由下述文件连接而成的库文件：

```
D:\aaa\yyy.obj 与 d:\bbb\zzz.lib
                        ;可能是存放在多个不同目录下的文件的组合
>tc xxx.cpp
                        ; tc 集成化环境。此时最好先加载鼠标驱动程序。
```

2. MS-VC(DOS/V 版 Visual C++ ver. 4.0)

(i) 进入 DOS 环境。

(ii) 安装系统后，在根目录下的 Autoexec.bat 中追加以下内容：

```
path %path%;c:\MS-VC\bin        ;MS-VC 是 Visual C++ 的目录
```

(iii) 使用 MS-VC(Microsoft Visual C++)时，需要执行存储在前述的\bin目录中的 msvcvars.bat 文件以进行环境设置，即：

```
>C:\MS-VC\bin\msvcvars.bat
```

(vi) 编译

```
>cl xxx.c               ;small 模式
>cl /AL yyy.c           ;large 模式
>cl /Alhd xxx.c         ;huge 模式
```

在连接几个模块时，可以先将其分别编译成 obj 文件(如 xxx.obj 和 yyy.obj 形式)，再用 link 命令将这些目标文件链接成一个扩展名为.exe 的可执行文件。

```
>cl /c/AL xxx.c          ;只进行编译，形成 xxx.obj
>link xxx yyy.obj zzz.lib ;链接，形成 xxx.exe 文件
>cl /AL xxx.cpp          ;编译.cpp 文件
```

此外，若编译时的错误提示信息较多，最初的信息可能因为屏幕的迅速滚动而不易看清，一种可行的办法是利用 more 命令暂停屏幕滚动：

```
>cl xxx.c | more
;"|"是 DOS 的过滤功能，此命令使信息显示满屏时暂停
```

3. VC98(Windows 的 DOS prompt VC++)

32 位的 Visual C++ ver.6.0 等系统安装后，除了 Visual studio 和 MFC 外，还会包括一个 VC98 目录。浏览一下可以发现，此目录下含有 bin 和 include 等目录，且\vc98\bin 目录还包括一个名为 vcvars32.bat 的文件、\vc98

\include 目录下存储着 iostream.h 和 stdio.h 等头文件。此时：

(i) 先创建一个批处理文件 vc32.bat，内容仅有一行：

D:\vc98\bin\vcvars32.bat ;假定 VC++ 安装在 D:\下[1]

(ii) 切换到 DOS 提示符。注意在 DOS 模式或 MS-DOS(16 位)时不能执行该程序。若系统出现了"Out of enviroment space"之类的提示信息，需要在 config.sys 文件中追加下述命令：

shell = C:\command.com /E:1280 /p

若 1280B 还不够大，可使用更大的缓冲区。

(iii) 重新执行 vc32.bat 时，若出现：

"Setting environment for using Miscrosoft Visual C++ tools"

信息，并能够正常执行 cl 命令，就可以编译 xx.c 和 xx.cpp 等程序了。

3.4.4 完全的 MS-DOS

有时，可能希望在一台 DOS/V 计算机上安装并使用 MS-DOS、Windows 和 Windows NT 等多种操作系统。这是因为，根据使用的控制板不同，如美国制的视频卡、DSP 卡等，可能要求机器必须能够支持 DOS/V 平台的 MS-C、Watcom-C、Texas Instruments 等不同的 C 语言环境。

◆ 几个 OS 共存——使用 SYSTEM Commander 4(Soft Board 公司)软件，该软件支持在 1 台计算机上共存几个 OS，附录 A 给出了一种在不同的 OS 之间进行切换的方法。

◆ HDD 替换——最近，出现了所谓支持 HDD 替换的计算机，不同的 OS 可以安装在不同的 HDD 上，用户可以指定使用任何一个 HDD 上的 OS，较为方便。

3.4.5 编辑和文件操作

这里就如何输入和编辑 C 语言程序稍做讨论。

若使用 DOS 环境，如 DOS 模式、MS-VC、Turbo C++ for DOS 或完全的 DOS/V 等，可以使用如下的方法输入和编辑 C 语言程序：

(1) 作为普通的编辑器[2](文本编辑器)，可从 DOS/V 用的 VZ 编辑器、Mifes ver.5.5 for DOS/V 等入手，也可以使用 Windows 版的 WZ 编辑器或 Mifes for Windows 在 Windows OS 环境下创建文本。

---

1) 由于 VC98 并不是存放在根目录而是 VC++ 的系统目录(缺省为 MsVStudio)，因此，步骤(i)中的批文件的内容最好写成 D:\ MsVStudio\vc98\bin\vcvars32.bat。——译者注

2) 原著此处所列出的编辑器在国内不常用，通常使用的是 QEdit、TC 或 BC(DOS 平台)以及 Windows 自带的记事本或写字板(Windows 平台)等。—— 译者注

(2) 使用DOS自身提供的编辑器。在DOS/V画面输入edit命令,系统启动编辑程序并切换到编辑窗口。不过,如果用户希望edit能够支持鼠标,则需要按如下方式事先加载鼠标驱动程序(按Alt+F结束edit):

① 通常,鼠标启动程序都会随机发布。修改autoexec.bat文件,将DOS用的PS2鼠标驱动程序追加上,使系统从Windows切换到DOS时可以查到该驱动程序。笔者的例子是在Autoexec.bat中追加path %path%; C:\msinput\mouse命令,这使PS2鼠标驱动程序可以通过path找到。于是,若切换到DOS画面,先输入"mouse"命令使鼠标驱动程序常驻内存,则在文本编辑器中就可以使用鼠标了。

② 执行DOS/V命令mouse/Z。

(3) 最简单的方法还是在Windows画面中将Windows版的WZ编辑器或Mifes for Windows打开,在此进行文本编辑,输入C语言程序。然后,切换到DOS提示符或DOS模式进行编译,检查错误,再返回到Windows进行文件操作。

(4) 使用前述的Visual Studio生成或编辑xxx.cpp。

(5) 使用Windows版的通用日文软件进行语言转换。

3.5 实现接口板控制的C语言程序

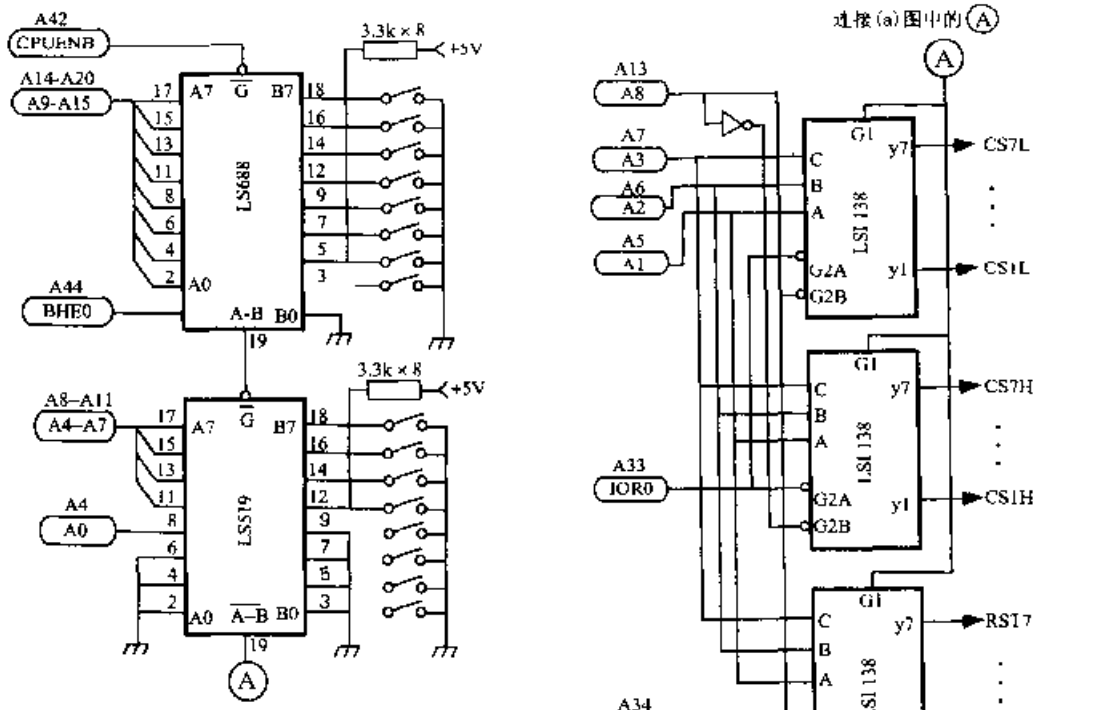
为了实现对机器人等机械设备的控制,需要随时把握设备的状态,为此必须输入适当的数据。或者说,对计算机和外设间的接口板的数据输入输出控制是必要的。从计算机的角度看,对机器人的控制,不过就是对接口板的控制。具有代表性的接口板包括计数板(称为编码板)、DA和AD转换板以及PIO板等等。通常,对这些接口板的输入输出控制方法与计算机的机种和OS有关。这里,我们将给出一个笔者自己编制的C语言程序示例,可以实现对PC-9800系列(C总线)所用计数板的存取和I/O控制。其中,OS采用MS-DOS,编译环境采用Turbo C++ ver.4.0。

应该说明,后续将有关于AD、DA接口的讨论,希望读者能够参考此处所给出的示例。

3.5.1 计数板的样式

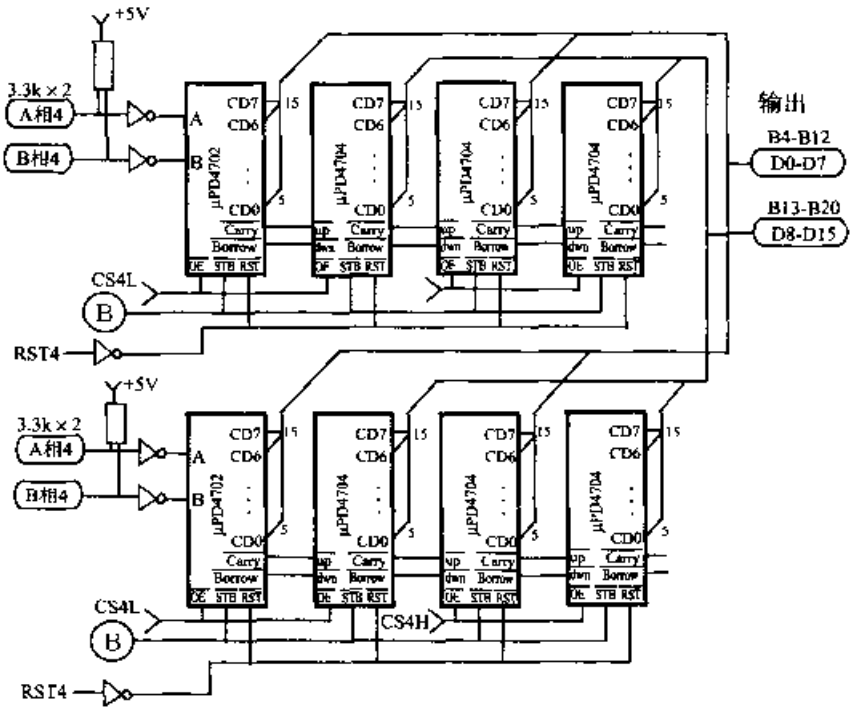
图3.1所示由笔者的研究室设计的计数板电路图。此板为32位的可逆计数器板(2个通道)。从旋转编码器输出的脉冲波的A相、B相作为输入,不论正转或反转都可以进行计数。

此板主要由地址译码部分、选择部分和计数部分构成,以下分别予以简要说明。



(a) 地址译码部分

(b) 选择部分



(c) 计数部分

图 3.1 32 位可逆计数板电路图(2 个通道部分)

1. 地址译码部分(74LS688、74LS519)

在机器人等的机械控制中,通常在1台计算机上要使用多个接口板,各板共同拥有地址总线和数据总线。因此,在计算机端,必须发出相应的指令信号才能选择一个接口板。对C总线所用的控制板来说,用户能够分派各板以不同的I/O地址,因此,可以从地址总线输出控制板的识别(选择)信号,利用地址译码器进行判别。

图3.2显示了输入给此控制板的数据结构。高12位中除去第8位(高)的剩余位用于I/O地址的识别,第8位(高)和低2～4位用于后述的选择部分,LSB总是0。必须注意的是,根据不同的OS,用户可用的I/O地址将受到限制,至于细节可参考对应的OS手册。

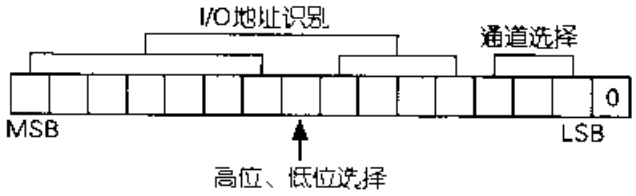


图3.2 输入给计数板的数据(CW)

2. 选择部分(74LS 138)

此选择部分包括选择输出数据高、低位的选择器及复位锁存器用的选择部分。

由图3.2可知,输入数据中,低2～4位的3位用于选择通道。而且,当CPU执行IN指令时(IOR0活动),选择输出数据高位、低位的选择器有效(变成活动状态),而当CPU执行OUT指令时(IOW0活动),复位和锁存用的选择部分有效(变成活动状态)。

其次,虽然此控制板是32位的,但将其用于PC9800系列(C总线)时,一次只能存取16位的数据。因此,32位的输出数据必须被分为高16位和低16位才能读入。使用此控制板时,利用输入数据高位的第8位来选择输出数据的高位和低位。即CPU执行IN指令时,若输入数据的高位的第8位为0时选择低16位,而高位的第8位为1时选择高16位,这就使得各个数据可以被正确读入。

此外,为了能够读取计数板的输出数据,必须对时刻变化的输出数据进行锁存。在此板中,当CPU执行OUT指令时,如果输入数据的低2～4位全部为0,可以对全部的通道进行锁存。而且,为了复位计数器,CPU执行OUT指令,用输入数据的低2～4位选择通道后,即可复位指定的通道。

3. 计数部分(4702、4704)

在此板中,使用4个8位的可逆计数器CMOS集成电路(4702×1、4704×3)构成了32位的计数电路。将从旋转编码器输出的脉冲波A相和B相(两者之间的相位差为90度)作为输入时,利用4702内的4倍频电路进行正转、反转

判定即可实现计数。当最低位的计数电路 4702 产生进位时，计数数据被逐个传送到高位的计数电路 4704。

3.5.2 输入控制程序

以下是此计数板的输入控制用 C 语言程序，其中的代码仅实现了对 2 个编码器的输出旋转角度数据的读取。

```
/* 计数板 I/O 地址 */
#define COUNTER1_LOW 0x00d2        /*计数器 1 通道的低 16 位数据
#define COUNTER1_HIGH 0x01d2       /*计数器 1 通道的高 16 位数据
#define COUNTER2_LOW 0x00d4        /*计数器 2 通道的低 16 位数据
#define COUNTER2_HIGH 0x01d4       /*计数器 2 通道的高 16 位数据
#define STROBE 0x00d0              /*锁存器

#include <stdio.h>
/* 为了实现计数器数据输入而定义的数据类型 */
struct enco{
        int low_data;              /*计数器的低位数据
        int high_data;             /*计数器的高位数据
        };
typedef struct enco ENCODATA;

union count_data{
       ENCODATA io_data;
       /*结构体 ENCODATA(2 个 int 型变量),与一个 long 型变量构成共用体
       ong long_data;
       /*使用 long_data 变量,可直接存取 32 位计数数据
       };
typedef union count_data COUNTDATA;

void main( )
{
  COUNTDATA count1, count2;        /*存放计数器 1、2 通道数据的变量
  outport(STROBE, 0);              /*输出传送给计数器板的锁存信号
  count1.io_data. ow_data = inport(COUNTER1_LOW);
                                   /*读入计数器 1 通道的低位数据
  count1.io_data.high_data = inport(COUNTER1_HIGH);
                                   /*读入计数器 1 通道的高位数据
  count2 io_data.low_data = inport(COUNTER1_LOW);
                                   /*读入计数器 2 通道的低位数据
  count2.io_data.high_data = inport(COUNTER1_HIGH);
```

```
                                        //读入计数器2通道的低位数据
    printf("COUNT1:%ldCOUNT2:%ld\n",      //输出long型整数
           count1.long_data, count2.long_data);
                                        //引用共用体COUNTDATA的long_data变量
}
```

根据前述讨论，程序中的32位计数数据分成高16位和低16位两部分，并分别用结构体ENCODATA的high_data和low_data成员将其作为int类型数据读入。其次，结构体ENCODATA(sizeof(int)×2=4B)和long型变量long_data(4B)构成共用体COUNTDATA，通过共用体中的long_data存取32位的计数数据。

也可以按下述方式进行操作：

```
ENCODATA io_data;
long long_data;
long_data = (long)io_data.high_data≪16 + (long)io data.low;
```

其中，高16位数据的int类型变量io_data.high_data被转换为long类型数据，即向左移16位，以添加低16位数据。对于共用体COUNTDATA则不需这种操作，处理也比较灵活。

在使用C语言编程时，需要注意如下一些常见的错误：

① C语言中有34个保留字，在定义类似free、long、auto和time等的变量时，注意不能使用保留字作为变量名，否则会引起冲突。

② 若感觉出现的错误有点“古怪”，可检查一下“{”和“}”的数目。或者，也可能是多写或遗漏了分号“;”。

如果编译时错误较多，可以考虑先修改前面的简单的错误，然后再运行程序试试。最后所剩下的含义不明的错误，通常与指针或数组有关。

③ 用scanf读取变量值时不要遗忘地址运算符&。

例如：scanf("%d", &a);

④ 对于数组，在用" "方式给字符数组赋初值时，数组的大小应该保证在字符个数+1以上。在C语言程序中，没有对数组边界的检查机制，因此，编译器不会意识到数组超界的错误，一旦使用了超出系统分配给数组的内存空间，可能使整个内存崩溃。

练习3

问题 3.1 按下述要求编制C语言程序。

(1) 在main函数中接收一个文件名，将具有1024个数据的数组t[1024]和x[1024]中的数据存储到该文件中。其中的t和x都是double类型的数据。

(2) 将数据存盘部分改写成独立的函数。

问题 3.2 编写满足下述要求的C语言函数。

(1) 从控制台接收字符串"123"，将其转换为整数123。

(2) 接收到字符串"ab01"即可生成文件名字符串"ab01.dat"。

(3) 一个文本文件b*.tab中含有一个内容为"1 23 sample"的行，编写一个只将数字部分转换为数值的函数。

问题 3.3 编写满足下述要求的C语言函数。

(1) 一直等待控制台的输入，若输入为Enter键则停止键盘接收。然后，将开头输入的数字字符转换为double类型的数值。若开头输入的字符不是数字字符则什么工作也不做。

(2) 内存中有表示时间的数组t[256]，将其数据都转换成字符串存储到x[256]中，并将数组(t_i,x_i)按列写入文件dat1.dat中。

问题 3.4 实现下述的位操作。

(1) 将从16位数据的LSB算起的第3位和第6位清0。

(2) 将从8位数据的LSB算起的第4位反转。

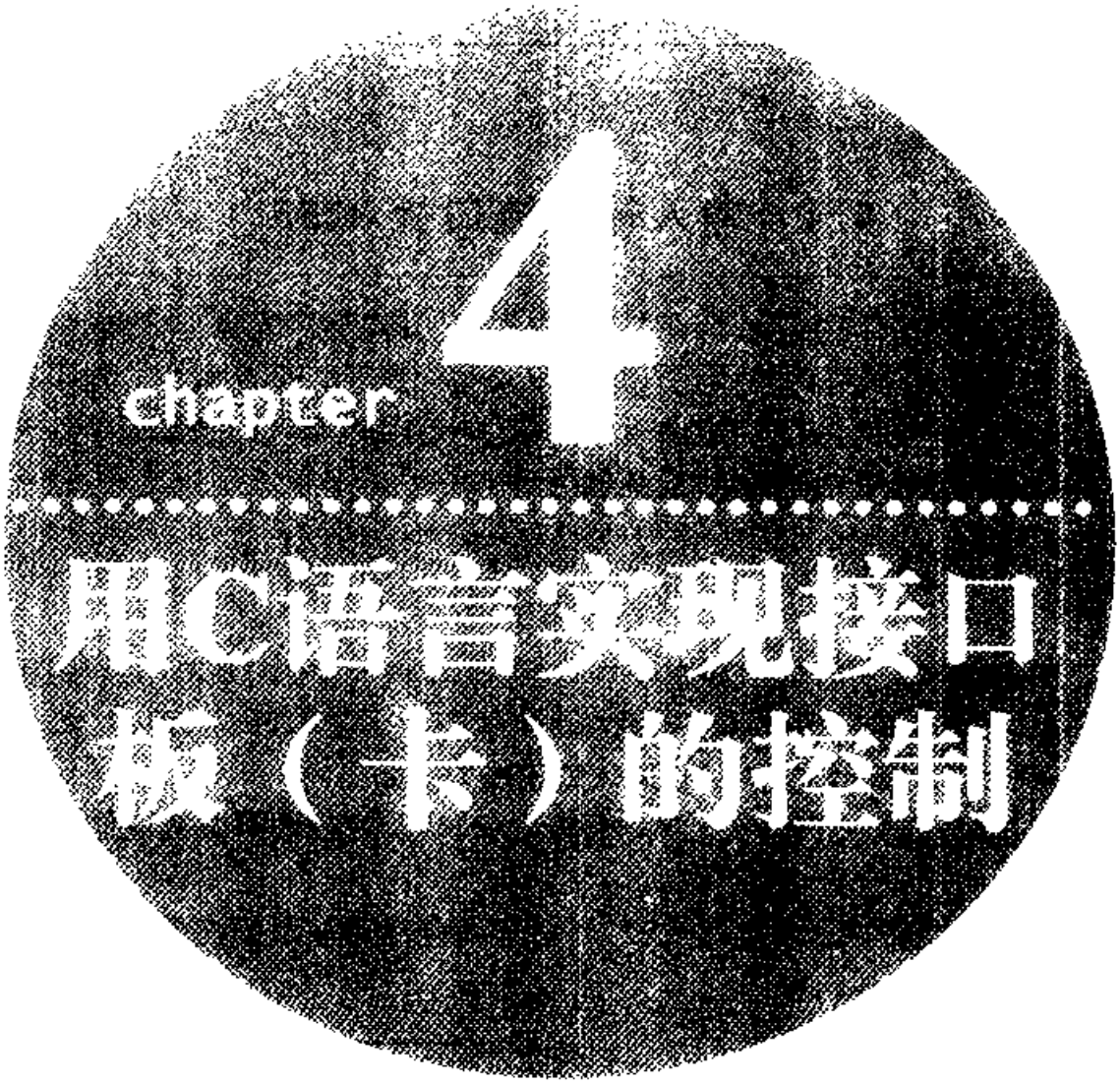
问题 3.5 编写计算矩阵A和矩阵B的乘积的C语言程序。

问题 3.6 编写满足如下要求的C语言程序。

(1) 输入的字符串中包含数字字符和其它字符，将其中的字符子串按字符串输出，数字子串转换为数值数据输出。

(2) 编写接收一个文件名的起始字符，即可生成如b01、b02、……的文件名的函数。

(3) 编写一个函数，该函数从main函数接收一个指针，通过该指针将z=x+y的运算结果返回给main函数。



在进行检测控制时，控制板通常插在计算机的扩展槽内，通过软件将其连接到微处理器。软件可以控制接口板LSI的启动、控制命令的写入以及数据的存取。出厂时制造商已对控制板进行了一定模式（使用方法）和命令（读写和中断等）设置。本章里，我们将学习有关基本接口板的使用方法、命令的书写方法，以及实际的控制程序的编写方法。

4.1 输入输出接口板 8255

在日本,通常这种接口板标有"并行输入输出接口板"的标记,市场售价也很便宜,购买时应注意其是适用于 DOS/V 类型还是 PC-9800 系列。对于具体的接口板来说,有通用的 PCI 总线型、DOS/V 专用的 ISA 总线型及 PC-9800 使用的 C 总线型等等,购买时也应注意区分[1)]。

4.1.1 8255

以下将对被称为 PPI(Programmable Peripheral Interface)的通用接口 8255 予以说明,此接口板的构成如图 4.1 所示[8,16]。在 8255 中,有 3 个 8 位的输入输出 IC,合计有 24 位的 TTL 输入输出能力。即

(1) 输入电压为 L(低)<0.8V、H(高)>2.0V。在使用开关控制时应注意防止振荡。

(2) 输出电压为 L<0.4V、H>2.4V。应特别注意在事前检查其是否有电流输出。

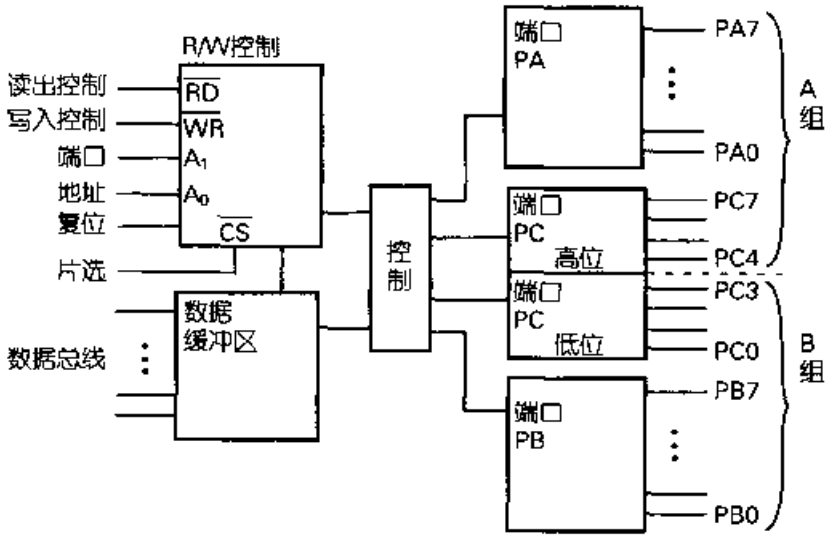


图 4.1 8255 的基本组成

1. 地 址

8255 占用 4 个地址,其中有 3 个与数据输入输出端口 A、B 和 C 相对应,分

1) 国内教材中通常称此接口芯片为 8255A。——译者注

别称为 PA、PB 和 PC。板地址由前述的地址译码器的设置决定。另一个地址是 CR(控制寄存器)地址,用于决定使用方式。

端口 A 的地址:PA——用户设定的偶数首地址

例如:0x0320

端口 B 的地址:PB——由 PA + 2 形成的地址(自动实现)

例如:0x0322

端口 C 的地址:PC——由 PA + 4 形成的地址(自动实现)

例如:0x0324

控制寄存器:CR——由 PA + 6 形成的地址(自动实现)

例如:0x0326

2. CPU 端的引脚说明

(1) 电源V_{cc}为 + 5V,GND 为 0V。

(2) 用数据总线 $D_0 \sim D_7$ 实现与 CPU 之间的数据存取。

(3) RD 和 WR 共用 CPU 的同一引脚,传输读写控制信号。

(4) A_1 和 A_0 是从 CPU 引出的总线,用于确定 8255 的端口地址。

(5) RESET 将系统恢复到初始状态。

(6) $\overline{CS}$为片选,将来自 CPU 的地址总线的信号解码并送出。

3. 机器连接端的引脚说明

(1) 端口 A 和端口 B 中,每个都可以单独指定为输入或输出。通常端口 A 作为输入而端口 B 作为输出,因为端口 B 的允许电流稍微大些。

(2) 端口 C 是 8 位的,也可以设定为输入或输出,此外,该端口的高 4 位和低 4 位可以分开使用,都可以单独指定为输入或输出。

4.1.2 关于工作方式

1. 方式 0

这是最常使用的方式,单纯地将所有端口都作为输入输出端口,从而形成了共 8 位 × 3 个的输入输出端口[17]。

2. 方式 1

称为选通输入输出方式。利用控制信号和状态信号使 CPU 和外部设备之间一边互相确认一边实现输入输出。使用端口 C 来为端口 A 和 B 提供控制信号和状态信号[16]。

3. 方式 2

只有端口 A 可以作为双向传输端口,端口 C 提供控制信号和状态信号。

以上为8255的3种工作方式,有关每种方式的详细说明可参考书后的参

考文献[25]。

4.1.3 控制字(CW)的设定

在8255中,CR为控制寄存器,用于端口选择等;CW为控制字,是传送给CR的控制字。

CR本身是可编程的,地址和工作方式等都可以由程序进行设置。CW的结构如表4.1所示[8,16]。其中:

(1) A组包括端口A和端口C的高4位。

(2) B组包括端口B和端口C的低4位。

(3) CW的设定

(i) $D_7=1$(MSB:最左端的位)表示此命令是CW(方式选择控制),而$D_7=0$表示此命令是位设置方式。

具体地说,CW有两种含义,其一是将端口分为2组,分别进行方式选择控制;其二是用于对C端口的每位置1或置0。一个具体的CW到底是哪种含义由D_7决定。

(ii) D_6、D_5用于A组的方式选择。

(iii) $D_4=0$表示端口A用作输出(output),$D_4=1$表示端口A用作输入(input)。

(iv) D_3的作用可参照表4.1。

表4.1 8255的控制字(CW)

CR	CW:功能	说 明
D_7	1:8255CW	控制字
D_6,D_5	00:方式0 01:方式1 1x:方式2	A组的模式 (x为1或0)
D_4	0(output),1(input)	端口A
D_3	0(output),1(input)	端口C高4位
D_2	0:方式0 1:方式1	B组的模式
D_1	0(output),1(input)	端口B
D_0	0(output),1(input)	端口C低4位

1. CW的例子

假定8255使用方式0,端口A(PA)作为输入,端口B(PB)和端口C(PC)作为输出,此时的CW如图4.2所示,组合后的结果见表4.2。

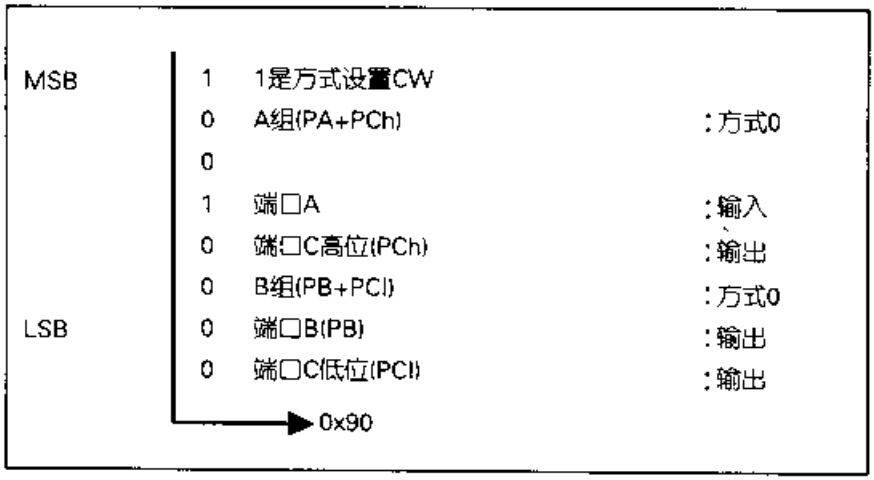


图 4.2 控制字示例

表 4.2 控制字(CW)

控制字 CW		PA	PCh	PB	PCl
10000000	0x80	out	out	out	out
10010000	0x90	in	out	out	out
10010001	0x91	in	out	out	in
10010010	0x92	in	out	in	out
10011000	0x98	in	in	out	out
10011011	0x9b	in	in	in	in

例题 4.1 假设板地址是 0x320,采用方式 0 工作的 CW 初始化为 0x98。利用 MS-C 编写读写控制程序。

❖解答❖ 板地址为 0x320 意味着 CR 为 0x326。初始化则是向 CR(0x326)输出 CW = 0x98。

```
outp(0x326, 0x98);          /* 初始化 CR */
d00 = inp(0x320);           /* 从端口 A 读数据并写入缓冲区 d00 */
outp(0x322, d00);           /* 向端口 B 输出 d00 的值 */
```

具有相同作用的汇编语言程序为:

```
MOV    AL,   98h    ;初始化 AL 寄存器为 98h
OUT    326h, AL     ;CR 的地址存入 326h
IN     AL,   320h   ;将端口 A 的数据读入到 AL 寄存器
OUT    322h, AL     ;同一数据写入端口 B
```

4.1.4 8255 的程序练习

将 LED 开关电路的接点状态读入 8255 的 PA,利用此接点状态控制与 PB

及PC相连接的LED阵列,使其发光(当然,也可以经过适当的运算加工后再输出)。假定使用正逻辑。为了使8255在高电平时能够输出较大的电流,必须加入相应的缓冲电路。图4.3使用了反相器7404作为缓冲器。

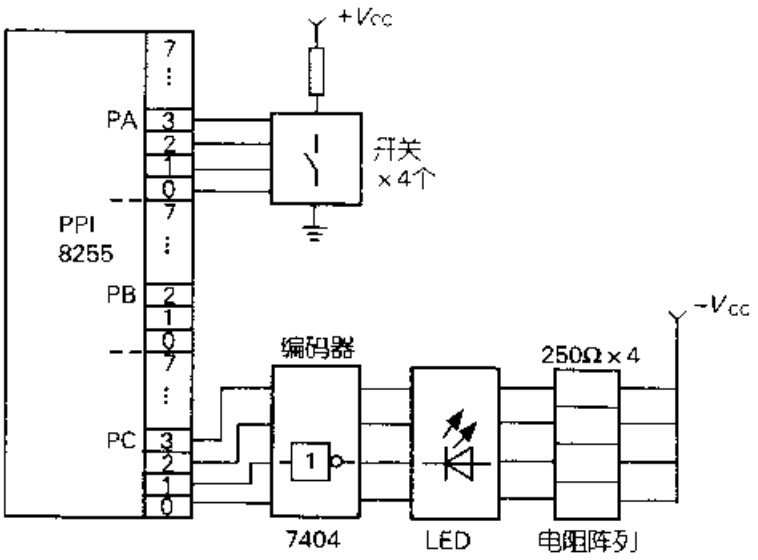


图4.3 LED控制器[例4]

1. 各端口的初始化和LED的发光控制

利用MS-C(或DOS/V版的MS Visual C++)编程时的代码如下:

```
/* switch and LED */
#include <stdio.h>          /* C语言特有的头文件 */                    ①
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#define PA 0x290            /* 8255的端口A地址 */                      ②
#define PB 0x292            /* 8255的端口B地址 */
#define PC 0x294            /* 8255的端口C地址 */
#define CR 0x296            /* 8255的CR地址 */
#define CW 0x90             /* CW:PA=in,PB=out,PC=out */               ③
void main( )                /* 主函数 */
{
  char a;
  outp(CR, CW);             /* 用CW初始化8255的CR */                   ④
  while(! kbhit( ))                                                    ⑤
  {
```

```
    d= inp(PA);          /* 从PA读取数据 */                    ⑥
    outp(PB, d);         /* 向PB输出d */                       ⑦
    outp(PC, d);         /* 在图4.3中只有与PC连接的LED */
    }                    /* while( )结束 */
}                        /* 主函数结束 */
```

以下是详细说明：

(1) C语言的标准头文件：

① <stdio.h>为标准输入输出等函数的声明文件

② <stdlib.h>为字符处理等函数的声明文件

③ <dos.h>为软件中断等函数的声明文件

④ <conio.h>为控制台等函数的声明文件。

(2) 用290h对8255的初始地址进行设置后，顺次地确定PA、PB、PC和CR的地址。为了合成16位的字，必须使用偶数地址。

(3) CW的设置方法如前所述。

(4) 使用8255时，必须先对CR进行初始化。利用函数outp( )实现至端口的输出。

(5) While( )语句的流程是：当括号内的条件为真时执行循环体{ }，否则结束循环。C语言的库函数kbhit( )的功能是测试键盘输入，若有按键则函数返回真。

(6) 函数inp( )的功能是从端口读入数据，并传送给变量d。

(7) PA作为开关，PB和PC作为LED，直接将开关缓冲区中的值原样输出到PB和PC。

2. 各端口的设置和数据的输入输出

使用汇编语言的程序代码如下：

```
;switch and LED
PA      EQU     290h
PB      EQU     292h
PC      EQU     294h
CR      EQU     296h
CW      EQU     90h
ORG     1000h
START:  MOV     AL,CW       ;AL是AX寄存器的低8位
        OUT     CR,AL       ;8255的初始化
LOOP1:  IN      AL,PA       ;数据输入
        OUT     PB,AL       ;数据输出
        JMP     LOOP1       ;循环返回
        END     START       ;程序终止
```

4.2 可编程定时器 8253

4.2.1 8253

PIT(Programmable Interval Timer)的结构如图 4.4 所示[16],右侧有 3 个彼此独立的 16 位的减法计数器。通常采用的计数频率包括:

8253——DC～2MHz

8253-2——DC～5MHz

8254——DC～8MHz

近来的 DOS/V 机中,比较常用的是具有 8MHz 计数频率的 8254。8253 的基本内容包括:

(1) CLK 端——此为时钟输入端。

(2) GATE 端——高电平开始记数,低电平停止。

(3) OUT 端——从计时器的计数值开始,根据 CLK 进行减法计数,到 0 时从 OUT 端输出。

(4) 有 6 种工作模式。

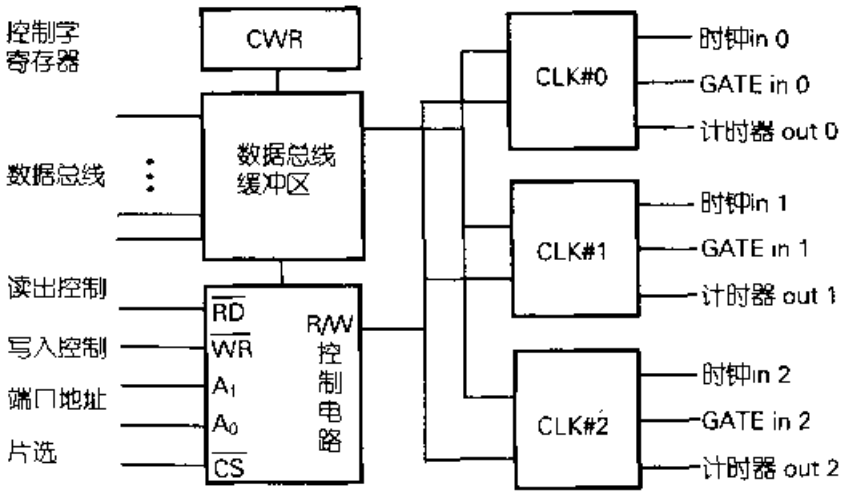


图 4.4 8253 的基本组成

4.2.2 8253 的工作模式

(1) 模式 0——单纯计数:当 CR 用 CW 设定为模式 0 后,OUT 变为 L 且在

到 0 之前一直维持在 L。设置好计数器的计数值后即开始进行减法计数。当计数器变为 0 时 OUT 端变为 H,计数停止。再次设置计数值则重新开始计数。计数完了即引发中断的场合使用此模式。

(2) 模式 1——可重复触发的单稳态触发器:GATE 变为 H 时,对计数器进行计数值设置后直接开始减法计数,此期间 OUT 端维持在 L。计数器为 0 时 OUT 端变为 H 并停止计数。当 GATE 再次从 L 变为 H 时又会重新进行减法计数,不必重新设置计数值。

(3) 模式 2——分频器:输入时钟被计数值 n 分频,加载计数值后开始计数。OUT 端在计数过程中一直保持 H,仅在计数即将停止的 1 个脉冲之前变为 L。占空比(duty 比,1 周期中 H 所占比例)等于发生的脉冲重复数与 n 的比。

(4) 模式 3——方波发生器:模式 3 与模式 2 的工作方式类似,但脉冲按模式 2 中占空比的 50%发生。计数值需要设置在 3 以上。因为每当计数结束时 OUT 端变为 L,故可用于区间计数。

计数值n是奇数时:脉冲的前一半共$(n+1)/2$个 H

脉冲的后一半共$(n-1)/2$个 L

(5) 模式 4——软件触发的选通信号发生器:当计数器减到 0 时,OUT 端变为 L 并维持一个时钟周期。与模式 2 不同的是,当写入新的计数值时,产生一个负脉冲作为选通信号,使计数器从头工作。

(6) 模式 5——软件触发的选通信号发生器:与模式 4 以类似方式工作,只是 GATE 变为 H 时重新进行计数器的初始化并开始新一轮的计数。

表 4.3 描述了这些工作模式。

表 4.3 8253 的工作模式[16,19,36]

模式	工作种类和计数开始	GATE	OUT
0	设置初始值后,直接开始计数	L:停止计数 H:开始计数	L:初始值和计数过程中 H:计数到 0 时
4	软件触发的选通信号发生器	同上	H:计数过程中 L:计数到 1 时
2	分频器	同上	按模式 4 的方式反复
3	方波发生器	同上	L 和 H 在结束过程中交替重复。 若计数值为奇数 H 多 1。
1	可重复触发的单稳态触发器	L→H 开始计数	L:初始值和计数过程中 H:计数到 0 时
5	软件触发的选通信号发生器	同上	与模式 4 相同

4.2.3 CW

8253的3个计数器彼此独立,各自按不同的方式工作。计数器的初始化

即是在CR中写入控制字CW,CW的格式如表4.4所示。

表4.4 8253的CW的格式[6,16]

CR	CW	功 能
D_7D_6	00	选择计数器#0
	01	选择计数器#1
	10	选择计数器#2
D_5D_4	00	计数器锁存(以便读出)
	01	只读/写低8位字节
	10	只读/写高8位字节
	11	先读/写低8位字节,然后再读/写高8位字节
$D_3D_2D_1$	000	模式0
	001	模式1
	x10	模式2
	x11	模式3
	100	模式4
	101	模式5
D_0	0	二进制计数
	1	BCD计数

1. CW的说明

(1) D_7D_8——选择计数:选择一个计数器。

(2) D_5D_4——加载数据:通常对16位数据先加载低8位字节,后加载高8位的字节。

(3) $D_3D_2D_1$——模式:选择前述的模式。

(4) D_0——数据类型:确定计数值是以二进制方式还是以BCD方式提供。

2. CW的设置

(1) 使用计数器#0时:$D_7D_6=00$

(2) 按先低位后高位的顺序读出、加载:$D_5D_4=11$

(3) 模式3:分频方波:$D_3D_2D_1=011$

(4) 二进制或BCD计数格式:$D_0=0$(二进制)

(BCD方式下直接以10进制值形式提供计数值)

综合以上情况,在进行二进制计数时,对应计数器#0的CW=00110110,即0x36,对应计数器#1的CW为0x76。若进行BCD计数,二者分别对应着CW=0x37和CW=0x77。

例题4.2 假定使用计数器#0进行计数,计数值按先低位字节再高位字节顺序进行设置,采用模式0,且计数类型为二进制类型。给出满足上述要求的相应的CW值。

❖解答❖ CW 的位组合形式为 00110000，即 0x30。

例题 4.3 假定利用计数器 #1 按模式 3 方式进行计数，计数值以二进制方式提供，按先低位字节后高位字节的顺序设置计数值。给出满足上述要求的相应的 CW 值。

❖解答❖ CW 为 0x76。

4.2.4 8253 的时钟设置

DOS/V 机用的控制板的时钟可按如下方法设置：

假定计算机端向 8253 输出的频率为：

$f_0 = 2.0\text{MHz}$

因为脉冲周期为：

$$\frac{1}{f_0} = 0.5 \cdot 10^{-6}$$

故 1ms 期间所需脉冲数为：

$$n = \frac{1 \cdot 10^{-3}}{0.5 \cdot 10^{-6}} = (2000)_{10} = (07d0)_{16}$$

而 10ms 期间所需脉冲数为：

$$n = (2000)_{10} = (4e20)_{16} \tag{4.1}$$

1. 计数器的初始值

设置计数器值为 1ms 时，按如下方式进行：

二进制方式的设置——低位 0xd0，接着高位 0x07。

BCD 方式的设置——低位 0x00，接着高位 0x20。

4.2.5 8253 的程序练习

此程序的假定条件是：时钟$f_0 = 2.0\text{MHz}$，使用计数器 #0，实现 1ms 计数。

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#define PIT0        0x300       /* 8253 的计数器 #0 的地址 */
//#define PIT1      0x302       /* 此处未用 */
//#define PIT2      0x304
#define PITCR       0x306       /* 8253 的 CR 地址 */
```

```
void init_8253(void)
{
    outp(PITCR, 0x36);          /* 8253的计数器#0、CR的初始化 */
    outp(PIT0, 0xd0);           /* 加载计数器低位1字节(式(4.1)) */
    outp(PIT0, 0x07);
}
    void main( )
{
    init_8253( );
    while(1);                   /* 无限循环 */
}
```

说明：

(1) 计数器设置：计数器#0；按低位、高位的顺序设置计数值；采用模式3；用二进制格式计数。若使用BCD格式则计数值为0x37。

(2) 若需要1ms以上的计数，只要替换为相应的数值即可。

(3) 无限循环中会产生定时中断，而中断产生后应如何处理将在下一章中说明。

4.2.6 任意时间的计数

以1ms为基准，任意指定计数值并使程序等待指定时间的wait函数是很常用的。下述函数的功能是：根据读入的一个整数d设置low_byte和high_byte，以实现d[ms]的计数。

```
unsihned int d;
unsigned char low, high;          /* 8位的低、高字节变量定义 */
                                  /* 将2B的整数拆成1B×2 */
void byte_data(unsigned int d, unsigned char * dlow, unsigned char * dhigh)
{
    * dlow = d & 0x00ff;
    * dhigh = d >> 8;
}
void main( )
{
    byte_data(d, &low, &high);
}
```

4.3 控制程序的应用

这里我们将根据实际的 I/O 板编制一个基本的控制程序。当然，需要先行了解所控制的设备才能进行控制程序设计，以便根据不同的控制对象采取不同的安全策略。

4.3.1 LED 和开关电路

此处使用的模型为 Sun Mitec 公司的教学用输入输出实验板(DOS/V 版的 Y98 控制系列)。这是一种 ISA 总线控制板，使用时插到计算机的扩展槽内。该板的组成包括图 4.1 中所示的 2 个 8255 和图 4.4 所示的 1 个定时器 8253，中断 IRQ 可以在 4 个中断请求中选一。由于 8253 和 8255 都是可编程 IC，使用之前必须先进行初始化。

1. 接 线

LED 及开关按第 2 章所述的接口电路中的连接方法进行连接。

(1) 8255 的 PA、PB 和 PC 全部(8 位×3 个端口)与 LED 连接。

(2) 8255 输出 1 即 H 时，经反相器反转为 L，使 LED 发光。

(3) 开关电路中直接使用触发开关，没加入防止振荡(电气噪声)的电路。

2. 8255 的使用条件

(1) 使用模式 0。对于模式 0，8 位×3 个端口全部用作输入输出端口。

(2) PA 作为开关输入，PB 和 PC 作为输出，此时 CW 为 0x90。

(3) 设置端口地址 PA = 0x80d0。此相当于提供了首地址 80h，DOS/V 机中即可正常工作。首地址直接确定了端口地址 PA，其后，PB、PC 和 CR 的地址可按序自动确定下来。

3. 时 钟

来自 ISA 总线的时钟具有 8MHz 频率，经 4 分频后变成 2MHz。

4. 中断请求

在 4 个中断类型 IRQ3、4、5 和 7 中可以任选其一。

5. C 语言的头文件

在通常的控制程序中，需要使用的 C 语言的头文件包括＜dos.h＞和＜conio.h＞。

4.3.2 控制 LED 循环发光[8,21]

1. 点亮与熄灭

基本的动作是通过定时器控制在一定时刻输出点亮信号,一定时刻输出熄灭信号,如此交替。至于定时器,既可以使用前述的硬件定时器,也可以使用此处介绍的软件定时器。

软件定时器:重复执行 printf("")语句 n 次,利用秒表可测出这个准确的次数。

2. 动态发光

在定时器的每次时间到时将输出数据做位移变换,以控制 LED 的点亮与熄灭,这种移位控制与加油站的店头广告原理相同。

3. 程 序

在 PA 的 PA0、PA1 和 PA2 上连接着开关。

(1) PA0 变成 ON 时,PB 的 LED 按位模式 11001100 每秒闪动 5 次。

(2) PA1 变成 ON 时,PB 的位模式每秒向右移 1 位,共执行 4 次循环位移。

(3) PA2 变成 ON 时结束。

循环右移是指,若移位前的 LSB(最右端的位)为 1,则移位后的 MSB(最左端的位)变为 1。

```
# include <stdio.h>
# include <dos.h>                    /* 控制程序中使用的C头文件 */
# include <conio.h>                  /* 同上 */
# define PA 0x80d0                   /* PPI-8255:PA 地址 */
# define PB 0x80d2                   /* PPI-8255:PB 地址 */
# define PC 0x80d4                   /* PPI-8255:PC 地址 */
# define CR 0x80d6                   /* PPI-8255:命令地址 */
# define CW 0x90                     /* PPI-8255:CW */
void wait(long int tt) ;             /* 函数 wait 的声明 */
void main( )
{
    int i, d0 ; char d1, d2 ;
    d1 = d2 = 0xcc;                  /* LED 的位模式 11001100 */
    outp(CR, CW) ;                   /* 8255 的初始化 */
z000: d0 = inp(PA) ;                 /* 从 PA 输入开关的位模式 */
    switch(d0)
    {
        case 0 : { goto  z000 ; }    /* 等待输入 */
```

```
        case 1 : { goto  z010 ; }          /* PA0 ON */
        case 2 : { goto  z020 ; }          /* PA1 ON */
        case 4 : { goto  z040 ; }          /* PA2 ON,结束 */
        default : goto z040;
    }
z010: for (i = 0 ; i<5; i++)
        {
            outp(PB, d1) ; wait (1000) ;   /* 发光 1 秒 */
            outp(PB, 0) ; wait(1000);      /* 熄灭 1 秒 */
        }
        goto z000;
    z020: for  (i = 0 ; i<5 ; i++)
        {
            outp(PB, d1) ; wait (1000) ;   /* 将数据 d1 输出到 PB */
            d1 = d1 >>1;                   /* 数据右移 1 位 */
            if(d2 & 0x01==1)
              d1 = d1 | 0x80;              /* 若 LSB 为 1 则 MSB 置 1 */
            else
              d1 = d1 & 0x7f;              /* 若 LSB 为 0 则 MSB 置 0 */
            d2 = d1;                       /* 保存值 */
    }
    goto z000;
z040:;                                     /* ";"是必要的 */
}
void wait( long int tt)                    /* 函数 wait 定义 */
{
    long int i;                            /* 定义 4B 大小的整数 */
    for(i=0; i<tt; i++)                    /* 假定 tt=1000 恰好为 1 秒 */
      printf(" ");
}
```

例题 4.4 若按下与 PA 相连的 8 个开关中的第 0 号和第 2 号开关，从 PA 得到的数据是多少？

解答 用语句 d = inp(PA);读取到的值 d 是 0x05。

4.3.3 用 PPI 控制电机

1. 匀速运转

为了在 SW 从 ON 到 OFF 的导通时间里，使齿轮电机匀速转动，我们需要控制其角度的位置，即实现所谓的 PWM(脉宽调制)控制。这里假定只有 1 个

转动轴。事实上,利用多轴时就可以构成多电机系统,实现对机器人的腕、肩和手等多部件的驱动。

2. SW 的设置

如图 4-5 所示,PA 为输入,PCH(高位 4 位)为输出,PCL(低位 4 位)为输入,此时有 CW = 0x91。将 PA 作为 4 个 SW 输入,有以下的数据对应关系:

(1) PA0　　00000001　　(0x01) ▸ 停止

(2) PA1　　00000010　　(0x02) ▸ 低速

/* 占空比为 1/3 的脉冲 */

(3) PA2　　00000100　　(0x04) ▸ 中速

/* 占空比为 5/10 的脉冲 */

(4) PA3　　00001000　　(0x08) ▸ 高速

/* 占空比为 1/1(连续波)的脉冲 */

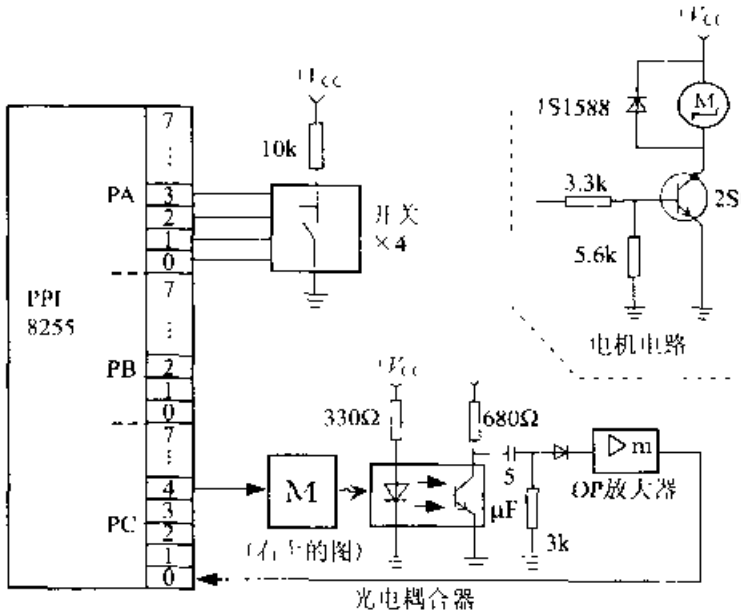


图 4-5　DC 电机的 PWM 驱动[18]

3. 软件定时器

转动速度使用软件定时器控制。在函数 wait 中假定 t = 1000 即为 1 秒。

4. 程　序

从 PC4 产生到电动机的输出,而光电耦合器的转动脉冲信号由 PC0 反馈到计算机。

下述程序只是给出了描述该过程的骨架。

```
void main( )
{
    int i, d;                                   /* PC0 的转动脉冲 */
    init_8255( );                               /* 初始化 */
    outp(PC, 0);                                /* 电机的初始设置 */
    while(1)                                    /* 无限循环 */
    {
        switch(inp(PA))                         /* 根据 PA 数据进行分支 */
        {
            case 1:outp(PC, 0);
                   break;
            case 2:                             /* 占空比为 1/9 */
                   outp(PC, 0xf0); wait(10);
                   outp(PC, 0); wait(90);
                   break;
            case 4:                             /* 0xf0 是将第 4 位变为 ON */
                   outp(PC, 0xf0); wait(50);
                   outp(PC, 0); wait(50);
                   break;
            case 8:
                   outp(PC, 0xf0);
                   break;
            default:                            /* 其它情况停止 */
                   outp(PC, 0);
                   break;
        }
    }
}
```

4.3.4 用 7 段 LED 元件表示数字

计算器的数码显示和移动电话的显示屏中经常使用 7 段 LED(由 7 个 LED 组成)。在图 4.6 中,LED 显示器的 7 段从 a 到 f 及 dp 分别与 8255 的 PB0～PB6 连接,PBi 为 H 时对应的 LED 元件发光。

(1) 若显示数字 1 则需要使 b(PB1)和 c(PB2)同时发光,即输出到 PB 的值是 0x06。

(2) 若显示数字 2 则需要使 a、b、g、e、d 同时发光,即输出到 PB 的值是 0x5b。

(3) 用 7 个 LED 元件表示字符时与上述方法类似。

(4) 简单地说,基本的操作步骤是,当计算机端输出一个数字或字符时,需要查询与之相应的 PB 端口数据表,即用 if-then 规则在表中检索应该输出到 PB 的数据。通常,使用译码器 IC 将对应的信号输出到 PB。

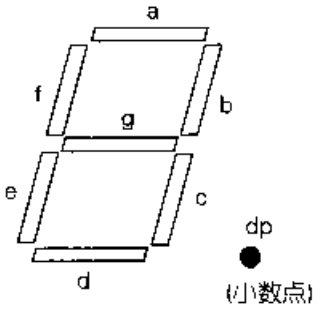


图 4.6 7 段 LED

练习 4

问题 4.1 用 C 语言编写满足下述要求的函数。

(1) 端口 A 的地址 PA 为 290h 时,从此端口读出数据并存储到(int)d0。

(2) 端口 B 的地址 PB 为 282h 时,从此端口读出数据并存储到(int)d1。

问题 4.2 按要求完成下述问题。

(1) 使用 PPI 的 8255,按下与端口 A 相连的第 0 号和第 3 号 SW 时,保证与端口 B 相连的 8 个 LED 中的第 0～2 号能够发光。请编制满足要求的 C 语言程序。

(2) 编制将(1)中的 LED 循环左移发光的程序。

问题 4.3 下述程序中使用了 PPI 的 8255。如图 4.7 所示,利用发光二极管(PD)的 ON 和 OFF 测量一种直线增长的物品的高度,光源来自位于 PD 对面的 LED。

(1) 假定板地址为 320h。

(2) 假定 PPI 的端口 A 为输入,端口 B 和端口 C 为输出。

(3) 当光被遮挡时 PD 为 ON。PD 与端口 A 的 PA0 端、PA1 端和 PA2 端相连。

(4) PD 的状态可以原封不动地由端口 PB0、PB1 和 PB2 输出,以控制暗壁的另外 3 个监视用的 LED。

(5) 端口 C 也与 LED 连接。

在下述程序的[ ]中填上适当的内容使程序完整。

```
/* switch and LED */
#include <stdio.h>
```

```
#include <stdlib.h>
#include <dos.h>
#define PA  (1)              /* 8255 port A address */
#define PB 0x0322            /* 8255 port B address */
#define PC 0x0324            /* 8255 port C address */
#define CR  (2)              /*  (3)  */
#define CW  (4)              /* CW:portPA = in,PB = out,PC = out */
void main( )
{
    char d;
    outp( (5) );             /* 用 CW 初始化 8255 的 CR */
    while(! kbhit( ))        /*  (6)  */
    {
        d =  (7)             /* 从 PA 读入数据 */
         (8) ;               /* 将 d 直接输出到 PB */
        if(d = =  (9) )      /* 如果 3 个 PD 都为 ON */
         (10)          ;     /* 使 PC 的 LED 全部发光 */
    }
}
```

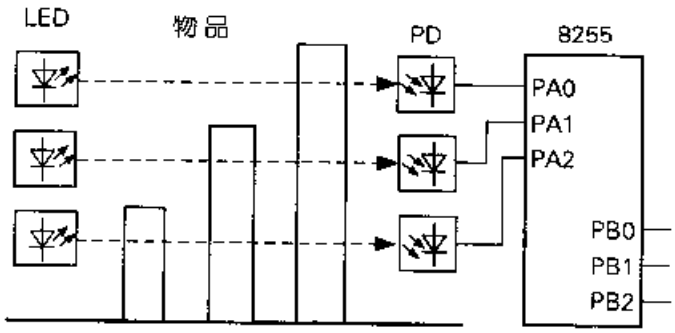


图 4.7　用 3 个透过型光电耦合器检测物品的高度

问题 4.4　假定设置在外部控制板上的 8253 具有 f_0 的基本频率。

(1) 若 f_0 = 2.4676MHz，制作使用计数器 #1、频率为 250Hz 的矩形波(采样频率数)。

(2) 若 f_0 = 2MHz，制作使用计数器 #0、频率为 500Hz 的矩形波(采样频率数)。

问题 4.5　假设作为输出的 PC 的 #0 与一个球盘的 SW(电磁开关)相连、#1 与一个转盘

的 SW 相连，而作为输入的 PA 上连接着开关按钮。

(1) 编制 C 语言程序，使其实现：若按下按钮 #0(PA0)使球盘工作，若按下按钮 #1(PA1)则使转盘工作。按下按钮 #2(PA2)时停止工作。

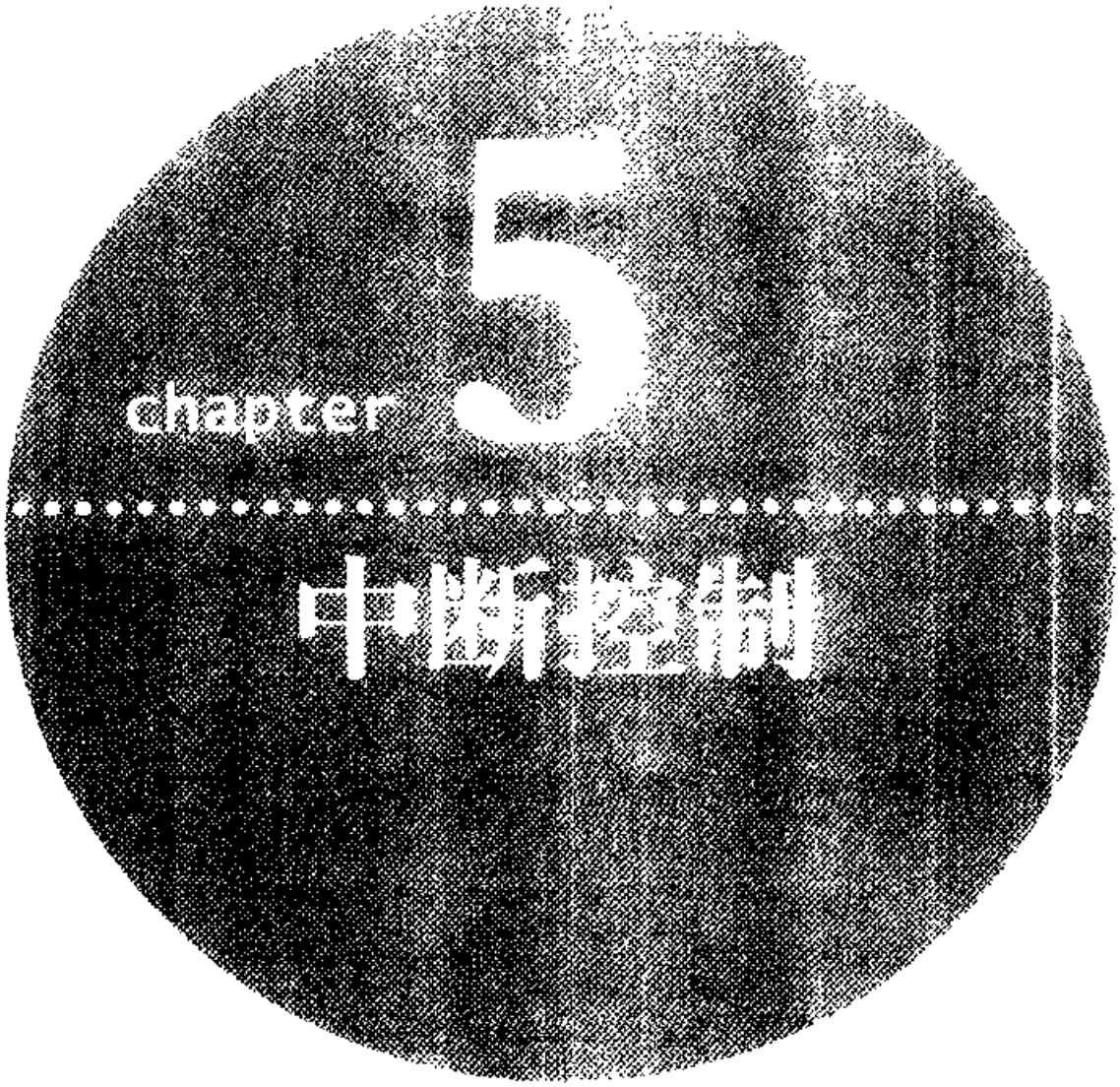
(2) 球盘和转盘中不论哪方工作，另一方即停止。

问题 4.6 利用 8255A 完成下述设计。

(1) 移动机器人的白色线条检测传感器

(2) 检测传送带上的物品通过数的计数器

问题 4.7 设计使用一个 7 段 LED，从计算机发出指令，使 LED 显示 16 进制数字 0～F 的数字电路并编制相应的 C 语言程序。



控制领域中的中断是不可避免的，而且计算机的中断也不像人们熟悉的电话的中断机制那么简单，而是十分复杂的，需要对计算机的硬件知识有较深的了解。但这种知识的学习毕竟不同于翻阅圣经，虽然你可以从阅读他人编写的程序中得到一些体会，但亲身实践才是掌握它的真正秘诀。开发中断程序时，应该将可执行程序复制到软盘上，然后在软盘上尝试执行该程序。坚持这种作法的原因是，基于磁盘不同的中断程序在执行过程中，有可能损坏磁盘文件。本章我们将学习8086系列（DOS/V机）的中断处理机制和中断程序的设计方法。

5.1 中断控制器

5.1.1 中 断

通常人们可以在正常业务处理过程中穿插处理别的事情,计算机也可以如此。计算机的中断(interrupt)可以按图 5.1[22]所示分为 3 类:

(1) 紧急情况下具有高优先级别的中断:如电源掉电等即属于优先执行的中断。

(2) 来自外部设备的中断,包括定时器和开关等。

(3) 来自软件内部的中断,称为 INT 指令。

(1)和(2)是**硬件中断**,(3)是**软件中断**,或称软中断。

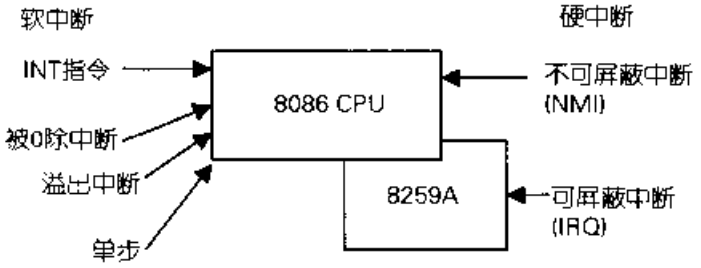


图 5.1 CPU 的中断源

1. 硬件中断

包括可屏蔽中断(即可通过编程禁止)和不可屏蔽中断二种。

(1) NMI——不可屏蔽中断称为 NMI,如电源掉电和内存的奇偶校验错误等。

(2) IRQ——可屏蔽中断,在 DOS/V 中称为 IRQ,PC-9800 系列中称为 INTR。通过编程存取**中断控制器** 8259A,可以允许或禁止对来自外部设备的中断请求的响应。系统中会对这些中断进行优先排队。

2. 软件中断

这是指使用 BIOS(Basic Input Output System,参照第 7.1 节的(a))的**系统调用**(INT 指令),如被 0 除和溢出等。事实上,CPU 频繁使用 INT 指令执行软件中断。

3. 中断处理顺序

(i) 外部设备提出中断请求。

(ii) CPU 接受中断请求并返回应答信号。

(iii) 临时中断正在执行中的程序。

(iv) 转移到中断所确定的地址。

(v) 执行中断程序。

(vi) 中断程序运行结束后，返回到原来的例程，继续执行被中断的程序。

5.1.2 CPU 和中断控制器(PIC:8259A)

Intel 8086 系列(包括 Pentium)的 CPU，通过可屏蔽中断 IRQ 对外部设备进行控制和响应。DOS/V 机和 PC-9800 系列机都使用 2 个级联的 8259A，采用**中断向量表方式**，16 个中断请求线中的 15 个留给外部中断使用，只有余下的 1 个用于连接第 2 个 8259A。

1. 任务分配

图 5.2[8,16]描述了中断响应和处理顺序：

(i) 外部设备向 8259A 发出中断请求。

(ii) 8259A 向 CPU 发出中断请求。

(iii) CPU 向 8259A 返回接受的中断请求信号。

(iv) 8259A 产生 1 个字节的中断向量 n，送入 CPU。

(v) CPU 参照在内存地址 0～1023 中存储的中断向量表中的($n \times 4$)地址，得到中断处理程序的首地址(指针)。首地址和内容可以由用户指定。

(vi) 先把将要返回到主程序的地址压栈(保存原地址)，然后执行指针所指向的处理例程。

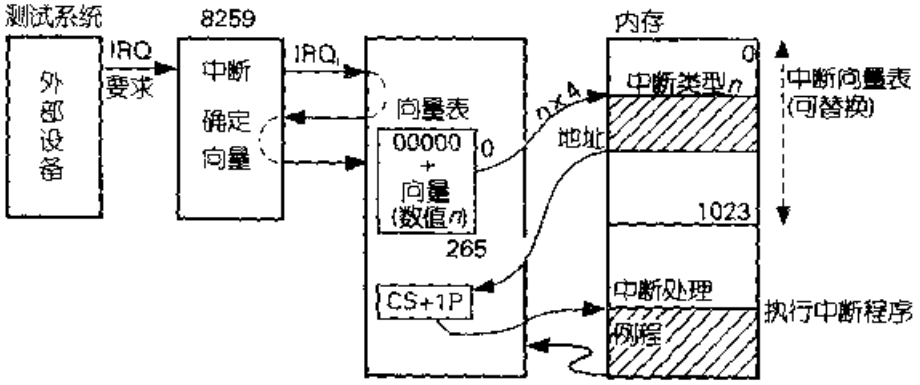


图 5.2 中断受理和处理顺序

2. 向量表

CPU(8086 系列)可以处理软件中断和硬件中断在内的 256 种中断,序号为 0～255。每个中断对应着相应的处理程序,而处理程序的首地址集中在一起,形成所谓的**向量表**,如表 5.1 所示。该表中的内容也称为**指针**。

(1) 内存地址中,保存有如下地址范围的 1024B 大小的**中断向量表**:

(段地址)0000:(偏移量)0000～03FF

(2) 表中的每个元素占 4 字节,用于记载中断处理程序首地址的段地址与偏移量。

(3) 发生中断时,根据此表的地址调用中断处理程序。

表 5.1 DOS/V 的中断(IRQx)控制器

Master (主)	向量	Slave (从)	向量	内 容		缺省的 板地址
IRQ0	08h			系统时钟		40h～43h
IRQ1	09h	—	—	键盘		60h
IRQ2	0Ah			级联到 IRQ9		
		IRQ8	70h	实时时钟		70h
		IRQ9	71h	级联到 INT 端		A0h～A1h
		IRQ10	72h	可能为空		
		IRQ11	73h	可能为空		
		IRQ12	74h	鼠标器	PS/2	
		IRQ13	75h	处理器	数值运算	0F0h～0F1h
		IRQ14	76h	硬盘	IDE1	1F0h～1F8h
		IRQ15	77h	可能为空	(IDE2)	170h
IRQ3	0Bh			串口 2	COM2	2F8h～2FFh
IRQ4	0Ch			串口 1	COM1	3F8h～3FFh
IRQ5	0Dh	—	—	并口 2	(LPT2)	278h～27Ah
IRQ6	0Eh			软盘	FDD	3F0h～3F7h
IRQ7	0Fh			并口 1	LPT1	378h～37Ah

5.1.3 DOS/V 的中断 IRQ[23]

1. IRQ(Interrupt ReQuest)

由于硬件中断请求线的限制,外部设备向 CPU 发出的硬件中断请求共分为 15 个优先级。

2. PIC(Program Interrupt Controller)

采用中断控制器 8259A,此功能集成在 CPU 芯片内。如表 5.1 所示,PIC 由主片(Master)和从片(Slave)构成,共同保存着中断向量(首地址)。

(1) 由于 DOS/V 中主 8259A 的 IRQ2 用于连接从 8259A 的 INT,从而实

现 2 个 8259A 的级联，不能连接外设，因此共可受理 15 个中断。即是说，IRQ2 与 IRQ9 是相同的，使用哪个都可以，通常使用 IRQ9。

(2) 按连接顺序，处于上方的中断具有较高的优先级。

表 5.1 中的 LPT1 用于连接打印机、CCD 照相机、CD-ROM 驱动器等外设，串口 COM1 上连接 LAN 卡。

3. 空的 IRQ

一般在 DOS/V 机中只空有 9、10、11 和 3(或者 5)这 4 个中断请求(可以由用户安排给附加的外设)，因此，当需要支持较多的控制板时必须避免 IRQ 的冲突(即不同设备使用同一个 IRQ)。不过，最近流行的 PCI 总线可以允许共享 IRQ(Share：若干个外部设备设置为同一个 IRQ)，因此，包括 IRQ9、10、11 和另外一个在内的 4 个中断请求基本是够用的。

4. 查询 IRQ 的方法

启动 Windows 后，在开始菜单中按如下方式(顺序)操作：

(i) 选择"控制面板"菜单项、双击"系统"图标、选择"设备管理器"页、用鼠标右键单击画面最上面的"计算机"项，系统会弹出对应的右键菜单，选择"属性"菜单项。

或者，也可以按下述方式操作：

(i′) 选择"控制面板"菜单项、双击"系统"图标、选择"设备管理器"页、展开"系统设备"分支、用鼠标右键单击某一项，在其右键某单中选择"属性"项，再选择"资源"页。

按上述方式打开窗口后，可以查看对应设备的中断请求号、I/O 地址和 DMA。

5.1.4 IRQ 冲突

若在 DOS/V 机中增加 2 个以上的插卡时发生了 IRQ 冲突，就必须改变其中之一的电路板上的 DIP 开关设置。此外，还要在 BIOS 设置程序中确认变更后 IRQ 的总线，若必要也需要作相应地调整。此时，可考虑使用如下方法：

(1) 在 Windows 中依赖 PNP 对 IRQ 自动确认。

(2) BIOS 设置——DOS/V 机内都设有 BIOS 设置程序，在 DOS 启动时会显示系统运行状态，通过键盘输入可以启动 BIOS 设置程序。

(3) 当一些重要的设备改变时需要在 BIOS 中进行重新设置，例如 HDD 变动、显示器(卡)变动以及修改了内存工作模式等。

总之，在增加或更改外部 I/O 板时还是很麻烦的。

5.1.5 地址冲突

商品扩展接口板通常都会占用一段连续的地址，实际使用时必须指出其首址。DOS/V 机中地址用 3 位的 16 进制数表示，应注意不要与表 5.1 中所列的系统使用地址和其它扩展接口板的 I/O 地址冲突。在 DOS/V 机中，重要的外部设备和扩展板的 I/O 地址分配范围是：

0x000～0x3ff

为用户保留使用的地址范围为：

0x100～0x16f、0x180～0x1ef、0x280～0x2af、0x300～0x31f、0x320～0x380

这些范围会因机器不同而有一定差异，最好先对自己的机器进行调查了解。

5.1.6 CPU 端处理中断的前期步骤

依据前述的 5.1.2 节中问题(1)的步骤(iv)和(v)，需要执行下述的一些步骤：

(i) 初始化 8259A

(ii) 关中断

(iii) 在内存的中断向量表中写入自己的中断程序地址

(iv) 将 EOI(end of interrupt，即中断结束)信号写入 8259A

(v) 开中断，执行处理程序。

系统允许用户重写中断向量表的内容。在主程序的执行过程中，若有来自设备的中断请求，CPU 会检查中断向量表，若遇到用户替换的指针则调用并执行用户的中断处理程序。

5.1.7 设置 8259A 的 ICW

设置初始化命令字(ICW：Initial Command Word)即是指对 8259A 进行初始化，它包括从 ICW1 到 ICW4 的 4 种命令字。

1. ICW 的设置

设置 ICW 时，必须按从 ICW1 到 ICW4 的顺序逐个进行设置。由于在计算机启动时会自动对 8259A 初始化，因此，通常我们不必再自己设置 ICW。

2. PIC 地址

DOS/V 机：主 8259A 的地址　　PIC0：0x20、0x21

　　从 8259A 的地址　　PIC1：0xA0、0xA1

例题 5.1 DOS/V 机的 IRQ2 和 IRQ9 是空的还是归用户使用?

❖解答❖ 用户只能使用 IRQ9。

5.1.8 设置 8259A 的 OCW

操作命令字(OCW:Operation Command Word)包括图 5.3[25,54]所示的 OCW1 到 OCW3 的 3 种命令。

(1) 在初始化后,通过使用命令字选择各种各样的操作:

IMR——设置中断屏蔽寄存器

ISR——确定中断服务中的中断优先级

IRR——确定中断请求中的中断优先级

EOI——通知中断结束并复位

(2) OCW 可在 ICW 之后的任何时候写入,以表示中断屏蔽和中断结束等(事实上,OCW 是在操作过程中给出的控制字,初始设定结束之后的命令都可以看作是 OCW 命令)。

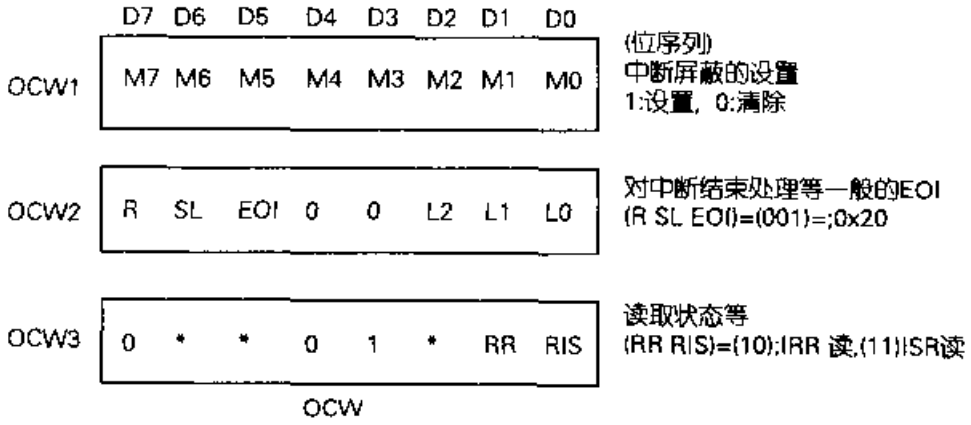


图 5.3 8259A 的 OCW 的设置示例[54]

1. OCW1

地址:PIC0 + 1(即 0x21)

(1) 也称此命令字为**中断屏蔽操作命令字**,用于设置中断的屏蔽和解除。

(2) 各位与表 5.1 中的中断请求输入 IRQ0～IRQ7、IRQ8～IRQ15 相对应。

(3) 位为"1"表示屏蔽,"0"表示解除屏蔽。

2. OCW2

地址:PIC0

(1) 改变中断优先级。

(2) 执行**中断结束处理**(EOI)[26]。对于**非指定 EOI**(或称一般的 EOI),用当前的中断优先次序(ISR 设置)清除最高的中断优先级。对于指定 EOI(或称特殊 EOI),清除指定的中断优先级。通常使用非指定 EOI 的 00100000(0x20)。

表 5.2 是一个 OCW 示例,可以参照表 5.1 和图 5.3 组成操作命令字。

(3) 向主 8259A 的中断发送 EOI 命令:

```
if(intn < 8)
    outp(PIC0, 0x20); // intn 为 IRQ 号
```

(4) 向从 8259A 的中断发送 EOI 命令时,因为要通过主 8259A,应按如下方式操作:

```
if(intn >= 9)
{
    outp(PIC0, 0x20);
    outp(PIC0, 0x0b);
    if(! inp(PIC0))
        outp(PIC0, 0x20);
}
```

表 5.2 OCW 的组成

OCW	地址	数据	16 进制	内 容
主 8259A 的 OCW1	PIC0 + 1	00000001	0x01	时钟中断设置(1 = set,0 = reset)
		00001000	0x08	IRQ3 或 INT0(PC-9800 系列)设置
		00100000	0x20	IRQ5 或 INT1(PC-9800 系列)设置
从 8259A 的 OCW1	PIC1 + 1	00000010	0x02	IRQ9 或 INT3(PC-9800 系列)设置
		00001000	0x08	IRQ11 设置
主 8259A 的 OCW2	PIC0	00100000	0x20	非指定 EOI

3. OCW3

地址:PIC0

(1) 读出状态,为 IRR 和 ISR 设置寄存器,以确定哪个寄存器的值将被读出。

例题 5.2 试给出 DOS/V 机中 IRQ3、IRQ5 的屏蔽设置数据(IMS)。

解答 分别为 0x80 和 0x20(参照表 5.1 和图 5.3)。

5.2 中断控制程序

5.2.1 中断基础

此处我们将给出一个描述下述过程的示例：先用 01010101 位模式控制 8255 的 PB 的 LED 之发光状态，当 PA 的开关变为 ON 时，产生 IRQ5 中断，使输出位模式经过了 8 位的循环移位，再经过一定时间后返回初始状态。例中使用了在 5.1.1 节中描述的有关 CPU 端处理中断的前期步骤。

1. 程序的条件

(1) 使用 IRQ5 作为开关：

对于主 8259A PIC 的 IRQ5，它的 IMS(屏蔽码)是 0x20，向量是 0x0d。

(2) 移位、发光等操作时，尽管可以使用以定时器中断为基础的 wait 函数，但为了安全此处使用了软件定时器。

(3) 软件定时器中需要使用 printf 函数，但由于 printf 函数本身利用了 BIOS 的软件中断，因此，软件定时器的前后需要经过屏蔽和解除屏蔽等处理，以避免输出中断与用户中断处理产生冲突。

2. 8259A(PIC)的初始设置

直接使用计算机启动时对 8259A 的初始设置。

5.2.2 中断定义函数

1. 基本的子函数

该函数使用 IRQ5 中断，在发生中断时 Iflag 被设置为 1 并返回到主函数。中断处理子函数应尽量简短。程序中需要使用与中断有关的一些头文件：

```
// intdef.h  for DOS/V
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#define PIC0        0x20        /* 8259A master address  */
#define PIC1        0xa0        /* 8259A slave address  */
#define IVEC        0x0d        /* 8259A master IRQ5 vector  */
#define IMASK       0x20        /* 8259A master IRQ5 set Mask  */
#define RMASK       0xdf        /* ~IMASK i.e. remove mask  */
```

```
int iflag,                                  /* interrupt flag */
void (_interrupt_far *Orgvec)();            /* original vector */
①void reinst(void)                          /* 重置函数:恢复原向量 */
{
    _dos_setvect(IVEC, Orgvec);             /* 用保存的 Orgvec 恢复 IVEC */
}
②void install(void (_interrupt_far *irqfunc)(), int intvec)
{
    _dos_setvect(intvec, irqfunc);
                        /* 将 intvec(向量)设置为 irqfunc(中断处理函数) */
    atexit(reinst);     /* 最后执行的函数,可放在任意位置 */
}
③void (_interrupt far IRQroutine(void)   /* 用户定义的中断函数 */
{
    iflag = 1;                              /* 执行时仅将中断标志置 1 即返回 */
    outp(PIC0, 0x20);                       /* 8259 的 EOI:非指定 */
}
④void init_int(void)                    /* 初始化 IRQx */
{
    _disable();                             /* 禁止硬件中断 */
    Orgvec = _dos_getvect(IVEC);            /* 保存原向量 */
    install(IRQroutine, IVEC);              /* 安装中断程序 */
    outp(PIC0 + 1, (inp(PIC0 + 1) & RMASK));        /* 8259A 中断使能 */
    _enable();                              /* 硬件中断使能 */
}
```

2. 说 明

_interrupt、_far、atexit 等是 MS-VC 的预定义标识符。

(1) 返回到向量 0x0d 被替换以前的状态,使计算机恢复正常工作的函数。为此,(4)的初始化之前需要用_dos_getvect( )将原来的向量保存在 Orgvec 中。

(2) 此函数是取得中断处理函数(3)的首地址(即 IRQroutine( ))和向量类型,并将其写入中断向量表的例程。实际上,它将中断函数的首地址写入 0x0d 的 4 倍的地址处。函数 atexit( )负责在程序结束时执行(1)中定义的 reinst( )函数。

· _interrupt- -用于定义中断函数

· _far 指针——超过 64kB 代码或者数据的程序必须使用 32 位的_far 类型指针。通常的 near 指针是 16 位的,代码段(CS)和数据段(DS)都是 64kB,但彼此分开。

① char _far func( );——此为返回 char 类型的具有 32 位地址的函数。

不管函数遇到 near 或是 far 地址均使用 32 位地址。

② char (_far *func)();——func 为指向返回 char 类型函数的 far 指针(32 位)。

(3) 此为用户定义的实际中断处理例程,用_interrupt far 作为中断函数的修饰。这里,函数只是对标志变量置 1 即返回。

语句 outp(PIC0, 0x20);是执行非指定 EOI(中断结束)。此例中用该语句结束对 IRQ5 的处理,使系统能够处理下一次到来的中断。

(4) main 函数中,先执行(4)以实现中断的初始化。

3. 8255 头文件

```
#include <dos.h>
#include <conio.h>
#define PA      0x300      /* 8255 port A addr  */
#define PB      0x302      /* 8255 port B addr  */
#define PC      0x304      /* 8255 port C addr  */
#define CR      0x306      /* 8255 CWR register  */
#define CW      0x90       /* CWdata:PA=in, PB=out, PC=out */

void init_8255(void)
{
    outp(CR, CW);
}
```

4. 中断主函数

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "intdef.h"                /* 包含中断的基本函数头文件 */
#include "l8255.h"                 /* 内容同上,二者结合使用 */
void waitx(int t);                 /* 时间等待函数 */

void main(void)
{
    int i, j;
    char d0, d1, d2;
    init_8255( );                  /* 初始化 8255 */
    init_int( );                   /* 中断初始化:前述的④ */
    d0 = d2 = 0x55;d1 = 0;         /* 发光数据 */
    while(! kbhit( ))
    {
      if(Iflag ! = 0)              /* 若发生中断,清除标志并转移到 z000 */
```

```
        }
            Iflag = 0;
            goto z000;
        }
        outp(PB, d0); waitx(1000);                    /* 交替输出数据 */
        outp(PB, d1); waitx(1000);
        goto z010;
z000:
        for(i=0; i< 8; i++)                           /* 发生中断 */
        {
            if(d2 < 0)
              { d2 = d2 << 1; d2 += 1; }              /* 位模式循环左移 */
            else
              d2 = d2 <<1;
        }
z010:;
        }
}
void waitx(int t)                                     /* 禁止中断,执行 printf */
{
        int i;
        outp(PIC0 + 1, inp(PIC0 + 1) | iMASK);        /* 禁止 8259A 中断 */
        for(i=0; i< t; i++)
            printf(" ");
        outp(PIC0 + 1, inp(PIC0 + 1) & RMASK);        /* 使能 8259A 中断 */
}
```


例题 5.3 计算位模式 10101010 循环左移 1 位的结果。

解答 若数据为负则左移后在最右位补 1,若数据为正仅左移即可。

5.3 再论测试板上的时钟

5.3.1 DOS/ V 扩展板上的定时器中断

这里我们将编制一个实现下述操作的程序:利用设置在扩展接口板上的定

时器引发中断，每隔 t(s)时间执行某项工作。例如，此工作可以是用 8255 使 LED 发光、发生矩形脉冲或控制电机的定时启停等等。

图 5.4 是前述的 DOS/V 版 Y98 控制板(扩展版 YVIO)[27]的时钟和定时器中断之接线图。YVIO 是以前的用于连接 I/O 板与 DOS/V 机的接口，具有 8 位的 ISA 总线。

1. 板上的定时器 8253，4 频分来自 ISA 总线的 8MHz 时钟，使之变成 2MHz。GATE 端有 10kΩ 的上拉电阻。

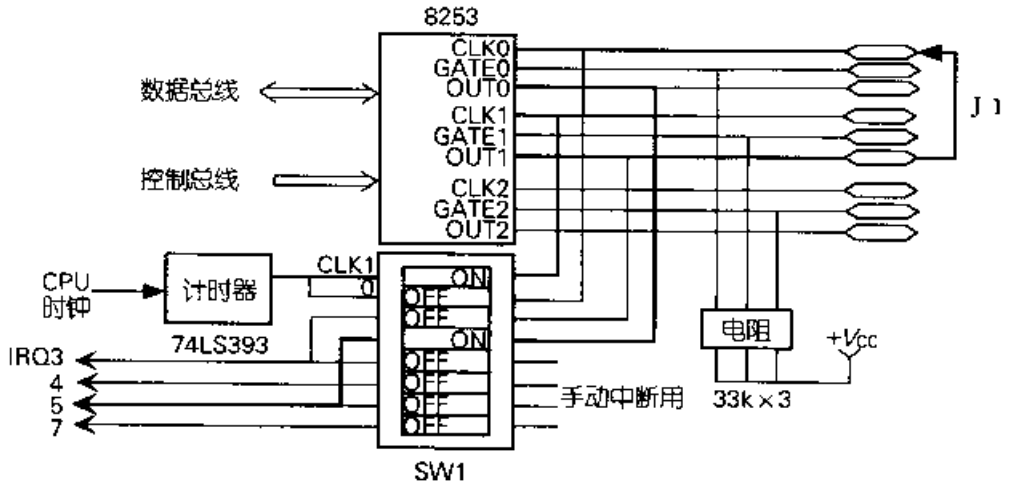


图 5.4 DOS/V 用扩展板(Sun Mytec 公司)的中断[27]

2. 中断请求

共有从 ISA 端口引出的 4 条中断请求线：

(中断请求)	(向量)	(屏蔽)	(DOS/V、I/O)
IRQ3	0bh	08h	COM2、串行
IRQ4	0ch	10h	COM1、串行
IRQ5	0dh	20h	LPT2、并行
IRQ/	0fh	80h	LPT1、并行

在 DOS/V 机中通常设有 2 个 ISA 端口，3 个 PCI 端口，可见，以上的 IRQ 中总会有空的。

3. 定时器的连接

假定在 IRQ5 为空的情况下，将控制板时钟 2MHz 用 CLK0 和 CLK1 进行 2 次分频后接入中断 IRQ5。此时，须根据如图 5.4 所示，用跳线 J1 将 CLK1 的 OUT1 端连接到 CLK0，CLK0 的 OUT0 端连接到 IRQ5。即有：

2MHz→CLK1，OUT1→CLK0，OUT0→IRQ5

4. 分　频

为了设置 1 秒间隔的定时器中断，首先在 CLK#1 将 2MHz 除以 1000，实现 0.5ms 的计数器。

$$\frac{2\text{MHz}}{1000} = 2000\text{Hz}\ \text{即分母的}(1000)_{10} = 0\text{x}03\text{e}8 \tag{5.1}$$

再在 CLK#0 将 2000Hz 除以 2000，实现 1s 的计数器。

$$\frac{2\ 000\ \text{Hz}}{2000} = 2\text{Hz}, \text{分母的}(2000)_{10} = 0\text{x}07\text{d}0 \tag{5.2}$$

5. 中断定义头文件

此处使用 IRQ5。

```
// intIRQ5.h     DOS/V 中断设置头文件
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#define PIC0          0x20          /* DOS/V PIC 8259A master address */
#define PIC1          0xa0          /* DOS/V PIC 8259A slave address  */
#define IVEC5         0x0d          /* PIC 8259A master IRQ5 vector   */
#define IMASK5        0x20          /* IRQ5 mask   */
#define RMASK5        0xdf          /* IRQ5 remove mask   */
int Iflag5;                         /* int flag1 */
void (_interrupt _far *org0d)( );   /*  RQ5 original vector */
①void reinst(void)                  /* 重置函数:恢复原向量 */
{
    _dos_setvect(IVEC, org0d);
}
②void install(void (_interrupt _far *irqfunc)(), int intvec)
{
    org0d = _dos_getvect(intvec);   /* 保存向量原始值 */
    _dos_setvect(intvec, irqfunc);
                                    /* 将 intvec(向量)设置为 irqfunc(中断处理函数) */
    atexit(reinst);                 /* 最后执行的函数,可放在任意位置 */
}
③void _interrupt far IRQ5routine(void)  /* IRQ5 中断函数 */
{
    Iflag5 = 1;                     /* 仅设置标志即结束中断 */
    outp(PIC0, 0x20);               /* 非指定 EOI */
{
④void init_IRQ5(void)               /* 初始化 IRQ5 */
```

```
}
    _disable();                                    /* 禁止硬件中断 */
    install(IRQ5routine, IVEC5);                   /* 安装中断程序 */
    outp(PIC0 + 1, inp(PIC0 + 1) & RMASK5);        /* 8259A 中断使能 */
    _enable();                                     /* 硬件中断使能 */
}
```

对①至④的解释可参见 5.2 节中对(1)和(4)的说明。

6. 扩展板定时器中断函数

```
// timer interrupt; DOS/V 扩展板
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include "intIRQ5.h"                    /* DOS/V 中断函数头文件 */
#define PA      0x300                   /* 8255 地址 */
#define PB      0x302
#define PC      0x304
#define CR      0x306
#define CW      0x80                    /* PA=输出, PB=输出, PC=输出 */
#define PIT0        0x310               /* 板上的 8253 计时器#0 地址 */
#define PIT1        0x312               /* 板上的 8253 计时器#1 地址 */
#define PIT2        0x314               /* 未用 */
#define PITCR       0x316               /* 计时器 CR 地址 */
⑤void i8255_i8253(void)
{
    outp(CR, CW);                       /* 初始化 8255 */
    outp(PITCR, 0x36);                  /* 8253:CLK#0,低位、高位,模式 3,二进制 */
    outp(PIT0, 0xd0);
    outp(PIT0, 0x07);                   /* CLK#0 计数值 0x07d0 */
    outp(PITCR, 0x76);                  /* 8253:CLK#1,低位、高位,模式 3,二进制 */
    outp(PIT1, 0xe8);
    outp(PIT1, 0x03);                   /* CLK#1 计数值 0x03e8 */
}
void main(void)
{
    i8255_i8253( );
    init_IRQ5( );                       /* 在头文件 intIRQ5.h 中定义 */
    while( ! kbhit( ))                  /* 等待中断 */
    {
    /* 其它处理 */
```

```
    }
    outp(PIC0 + 1, inp(PIC0 + 1)  MASK5);          /* 中断禁止 */
}
```

例题 5.4 用 BCD 码实现上述程序中的⑤。

解答 代码如下：

```
outp(PITCR, 0x37);          /* 8253:CLK#0,低位、高位,模式 3,BCD */
outp(PITCR, 0x77);          /* 8253:CLK#7,低位、高位,模式 3,BCD */
outp(PIT0, 0x00);
outp(PIT1, 0x10);           /* CLK#1 计数值 1000;0.5ms */
outp(PIT0, 0xC0);
outp(PIT0, 0x20);           /* CLK#0 计数值 2000;1s */
```

5.4 利用 CPU 系统定时器的中断实现

计算机的系统时钟总包含有专用的中断接口，DOS/V 或 PC-9800 系列都如此，它们使用预先设置好的特定端口。

5.4.1 DOS/V 系列的系统定时器[28,29]

在 DOS/V 机中都有系统定时器 IRQ0 和实时定时器时钟(RTC)IRQ8。DOS 通过第 7 章表 7.1 所示的软件中断的 INT 1Ah 实现对二者的支持。

1. 系统定时器

图 5.5 所示的 IC8254(8253 系列)有 3 个可编程定时器。只能使用定时器 #0 实现 IRQ0。其基本设置如下：

基本频率：f_0 = 1.19318MHz[29]

通常用计数值 65536(= 10000h)进行分频后，有：

频率：18.2(精确值为 18.206482)Hz

周期：55(精确值为 54.9255)ms

此定时器的输出端 OUT #0 与 PIC8259 的定时器 IRQ0 相连，可以定期产生中断。且有

分频值：$\frac{65536}{2^n}$

对每次中断都必须发送 EOI(0x20)控制字，清 8259。

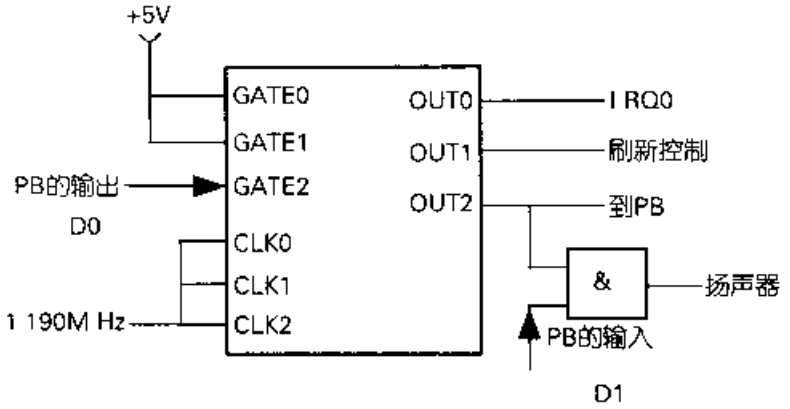


图 5.5 DOS/V 系列的系统定时器(8254)的结构[28]

2. 系统定时器 BIOS[28,29]

读时刻计数值	写时刻计数值
定时器:AH 寄存器 = 0x00	定时器:AH 寄存器 = 0x01
INT 1Ah	INT 1Ah
输出: CX 寄存器 = 计时器高位值	输入:CX 寄存器 = 计时器高位值
DX 寄存器 = 计时器低位值	DX 寄存器 = 计时器低位值

3. 实时定时器时钟(RTC)

定时器时钟使用 IRQ8,基本频率和分频值与系统定时器相同。BIOS 的设置为:

读实时定时器时刻:AH 寄存器 = 0x20
写实时定时器时刻:AH 寄存器 = 0x30
定时器 BIOS:INT 1Ah
输入输出:CH 寄存器 = 时(BCD 码)
CL 寄存器 = 分(同上)
DH 寄存器 = 秒(同上)

4. C 语言的 BIOS 调用

```
union REGS inregs, outregs;
                                    /* 寄存器共用体 inregs 和 outregs 定义 */
inregs.h.ah = 0;                                    /* 读计数值 */
int86(0x1a, &inregs, &outregs);                     /* BIOS 调用 */
```

5. 程序示例

请参考附录 C。

例题 5.5 假定定时器 IC(8254)的振荡频率是 2MHz,试制作周期为 10μs 的计数器。

❖解答❖ 由定时器时钟 2000kHz,即 0.5μs 周期,可知(分频时钟的分母(devisor))为:

$\frac{2000\text{kHz}}{20} = 100\text{kHz}$,周期为 10μs。

练习 5

问题 5.1 请给出 IRQ3、5、10 的向量和屏蔽数据。

问题 5.2 请按步给出下述 C 语言程序的说明,其中,intn 是 IRQ 号,msk 是屏蔽数据。

```
void irq_set(int intn, int msk, void (_interrupt _far * intfun)(void))
{
  _disable( );
  Orgvec = _dos_getvect(0x0b);
  _dos_setvect(0x0b, intfun);
  _enable( );
  outp(0x21, inp(0x21) & ~msk);
}
```

问题 5.3 试给出 8259A 的 IRQ3 中断的中断结束命令。

问题 5.4 暂停定时器,然后输出提示,再重新开放定时器,应如何处理?

问题 5.5 假定 DOS/V 扩展板的定时器时钟 f_0 为 8MHz。试制作用定时器 CLK#0 和 CLK#1 进行分频,构成间隔为 1ms 的中断定时器。

问题 5.6 有一块 ISA 总线板,根据说明书中的说明,可以使用 IRQ10、11、12 或 15 中的任何一个中断请求,但实际设置并测试后发现哪个也行不通。通过 Windows 的"控制面板"中的"系统"工具查看,发现 IRQ10 为声音设备占用,而其它的中断分别被鼠标器、数值数据处理器和 Internet 接口板占用,都没空闲。试问此时应如何处理?

chapter 6

AD和DA接口

数字化技术已渗透到社会生活的各个方面，甚至造成了什么都可以用数字来处理的错觉，而自然现象和我们身边的事情几乎都是模拟的。确实，数字技术作为计算机的核心技术渗透于检测、控制和通信等各方面，使精度、速度和可靠性都有所提高，也适应了自然界和数字机器相连接时需要进行中间信号处理的需要。例如，数字电话等设备，先将模拟量（声音）转换为数字量（AD变换），最后再将数字量转换为模拟量（DA转换）。

在进行检测控制时，这类控制板被插在计算机的扩展槽中，通过软件连接，与计算机进行沟通。通常在出厂前，制造商已在控制板上做了特有的设置（使用方法）并提供了必要的命令。本章中我们将从AD和DA控制板的角度学习其使用方法和控制命令的生成方法。

6.1 AD转换和前处理

6.1.1 模拟量控制系统的组成

关于AD和DA转换器的作用正如第1章所做的介绍，利用传感器测量出温度、压力、光和声音等模拟量，经AD转换器将其数字化，再送入计算机进行计算、判断等处理，最后经DA转换器转换为模拟量输出，以驱动电机等执行装置。与PPI的8255不同，AD和DA转换器以大约－10V～＋10V的连续信号[30]作为处理对象。

6.1.2 AD转换

近来的AD转换器(ADC：Analog to Digital Converter)较以前有很大变化，不仅工作速度有较大提高，应用范围也不仅限于直流检测过程，音频和视频领域的交流信号处理中也充分利用了AD转换器。在处理交流信号时，要求对电压和频率2种量进行正确的变换。

1. 采　样

为了实现模拟量到数字量的转换，必须进行模拟量的样本化(采样，Simpling)、量化(数字化：Digital)和符号化(编码化：Coding)，如图6.1所示。其中：

(1) 采样电路是将模拟信号的瞬时值每隔一定周期进行记录的电路，为此需要使用专门的IC。

(2) 量化是指将模拟量转换为数字量。

(3) 符号化是将数字量转化为2进制n个位的数据。

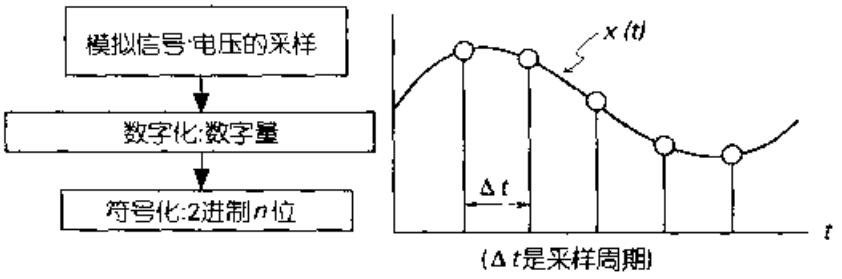


图6.1　AD转换过程

2. 采样定理

为了能够用采样得到的信号完全再现原信号，用于采样的采样频率f_s必须在原信号波源的最高频率f_m的2倍以上，即必须满足：

$$f_s \geqslant 2f_m \tag{6.1}$$

此即所谓的采样定理，其中的f_s称为尼奎斯特(Nyquist)频率。即当信号$x(t)$的频率不超过f_m时，$x(t)$可由依据采样频率f_s采样所得的采样值完全恢复，这意味着，在连续的数据处理中不会产生自身干扰。

3. 量化误差

变换后的曲线中，最低有效位LSB(least significant bit)呈阶梯状，自然会产生LSB的±1/2倍范围的量化误差。2进制数的1位称为**比特**(bit:binary digit)。分辨率(或称分解度)可由下式给出：

$$\frac{\text{满量程电压}}{\text{用比特数表示的最大数值}+1} \tag{6.2}$$

引起AD、DA转换误差的原因很多，可根据转换的比特数决定转换精度。转换比特数增加时转换所需要的时间也相应增加。

6.1.3 滤 波

由于AD板的能力决定了采样频率f_s的上限，也决定了满足式(6.1)的检测对象的频率上限f_m。相应地，当检测对象所处频率较低时，f_s也应较低。应注意检测对象不能包含超过f_m的频率，因此，需要进行下述滤波过程。

1. 反走样(anti-aliasing)滤波

利用满足采样定理的最大频率f_m过滤掉峰值的低通滤波器(LPF)，即过滤掉输入频率中含有的不需要的高频率部分。

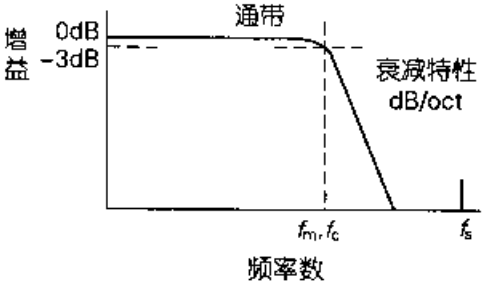


图6.2 低通滤波

2. 滤波特性

通常，图6.2的通带部分使用平坦的巴特沃斯特性中的直线相频特性，而

截止特性使用具有 18dB/oct(60dB/dec)以上的斜率特性。

3. 采样频率

为了提高精度,实际使用的采样频率f_s通常为f_m的 3～5 倍以上。例如,对于不超过 3.2kHz 的电话使用 8kHz、不超过 4.2MHz 的 TV 采用 14.3MHz、不超过 20kHz 的 CD 采用 44.1kHz 的频率进行采样。

例题 6.1 发动机的振动频率最高为 200Hz,请给出必要的采样频率和过滤器。

解答 使用截止频率f_c = 200Hz,斜率特性在 18dB/oct 左右的 LPF,采样频率f_s = 1kHz 左右。

6.1.4 输入方式

1. 单端(single end)输入

如图 6.3(a)所示的方式[31]。由于信号源和 AD 板间的感应噪声会导致电位差的产生,从而形成误差,因此,使用这种方式时应该在周围没有其它电气设备和电子机械的状态下进行检测。

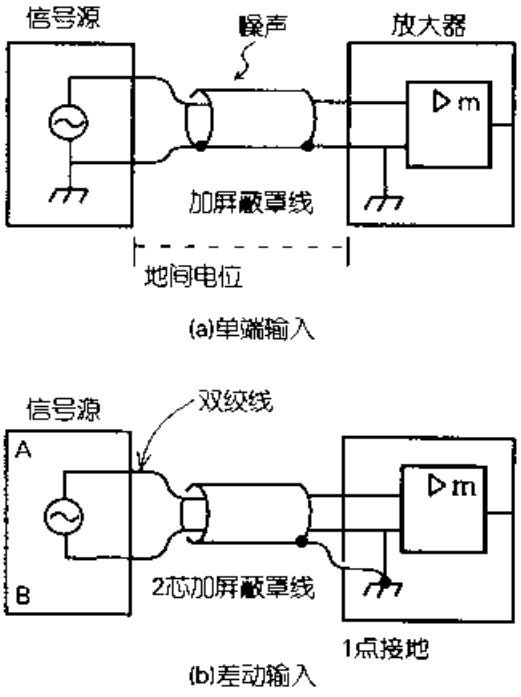


图 6.3 信号源的连接方法[32]

2. 差动输入

原理如图6.3(b)所示，此方法利用基准电位浮动到A点和B点的差来计算电位差。在2芯屏蔽线和信号源内部，采用双绞线(图中未画出)的方法来降低噪声。屏蔽罩与信号源的地相接(1点接地)，使二者具有相同的地电压。

差动输入方法不受噪声和地间电位差的影响，即使周围有噪声源也可以达到较高的测量精度。

6.2 AD转换器

6.2.1 转换方法

转换速度是指从发出转换指令开始，到形成数字数据输出为止所需要的时间。

如图6.4所示，主要有以下的转换方法和特点：

(1) 积分型转换的特点是速度低但精度高；逐次比较型转换具有中等速度和中等精度；并联(并行)型转换则具有较高速度，但精度较低。

(2) 音频信号具有较宽范围的动态变化频率，在高级音响中要采用16～20位、变换速率在48 ksps的Δ-Σ比较型及串并联型转换。

(3) 视频信号通常采用10～20位并具有10M～30Msps以上转换速度的全并联型专用IC。

这里的sps表示"sample/s"之意。电话等设备比较强调数据传输量，故通常使用8位的转换，而检测中由于要求有较高的精度而使用12～16位的转换。

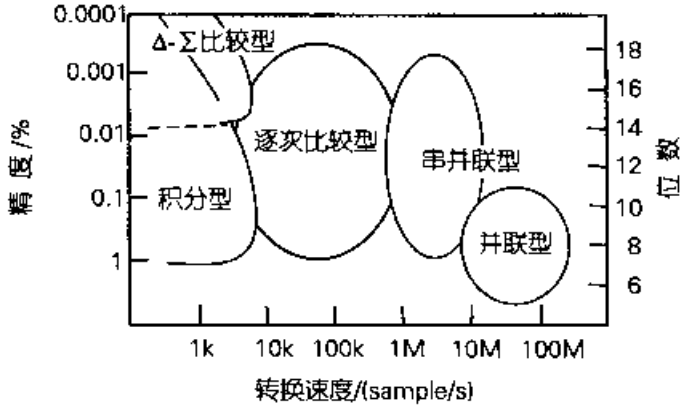


图6.4 AD转换的方式和精度

例题 6.2 试计算转换速度为 5.6μs，±0.5V 的满量程电压转换为 12 位的 AD 转换的精度。

❖解答❖ 根据式(6.2)，12 位的数值范围为 4096 + 1。因此，转换精度为：

$$\frac{1V}{4096} = 0.000244(0.0244\%)$$

6.2.2 采样-保持的必要性

在检测控制过程中广泛使用逐次转换式的 AD 转换器，很明显，若在转换过程中输入数值有 1LSB 以上的变化就不能进行正确的转换。因此，要附加一个所谓的采样-保持(S&H)电路，以使转换中的输入数据能够保持一定时间。

在采样 保持电路中要使用很多专用的 IC，因为多路数据需要逐个进行转换，必须精确地取到同一时刻的多路数据。近来，并行的 AD 转换方式已普遍使用，主流产品是支持 4 路同时转换的 AD 转换器。

6.2.3 AD 转换电路

1. 积分型 AD 转换器

图 6.5 所示为积分式 AD 转换器电路，这种转换方法的优点是精度高、抗干扰能力强且价格便宜，缺点是转换速度较低。双积分型 AD 转换器由积分器、比较器、基准电压、计数器和开关组成。

(i) 开始时将模拟电压v_1进行一定时间的积分，在积分电容C中存储电荷。

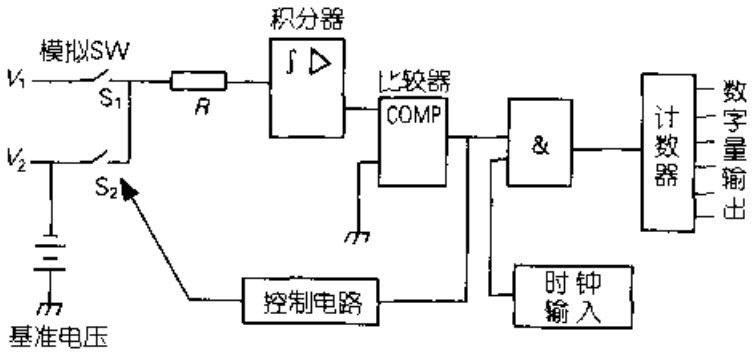
(ii) 在基准电压v_2下再次进行积分(反向积分)，使电容放电。

(iii) 利用比较器进行检测，直到积分器的输出为 0，在此期间利用计数器对所需时间进行计数即可得到数字量输出数据。

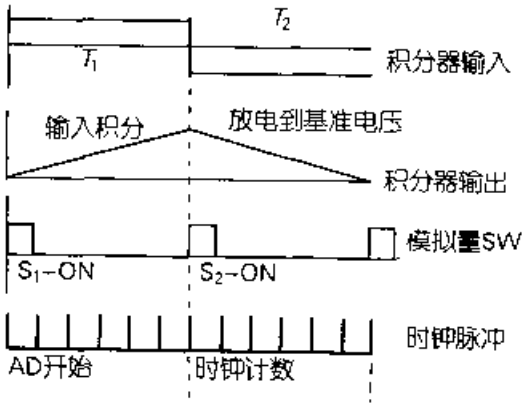
因为电荷的积蓄和放电需要时间，导致变换时间会有毫秒(ms)级的延迟。但由于转换精度较高，故可用于温度检测等方面。

2. 八-Σ 型转换方式

在音频领域中使用 Over Sampling 方式(△-Σ 型 AD 转换器)。基本原理是，先对模拟信号进行积分，并对其输出进行 AD 转换。再将其用 DA 转换还原为模拟量，向积分器反馈。控制积分器的误差为 0，即可以得到正确的数字量输出。采用此种变换具有很高的精度，故多用于音响设备。



(a)结构图



(b)时 序

图 6.5 积分式 AD 转换器

3. 逐次比较型 AD 转换器

逐次比较型 AD 转换器由基准电压、比较器、DA 转换器和比较电路构成，工作原理与天平相同。即在测量未知重量的物体时，通过逐渐增加或减少砝码的方法使其平衡。图 6.6 是最普通的逐次转换式电路。此种转换具有较好的精度价格比，转换速度在微秒(μS)级，输出信号位数为 8～12 位。

在逐次比较型 AD 转换器中，天平是比较器，砝码是 DA 转换器，砝码的选择来自逐次比较逻辑[33]。工作原理是：

(i) 首先将最高位置 1。将所有位构成 2^n 作为输出数据，送入比较器与输入值进行比较。

(ii) 若输入值小则减去重量 2^n-1，若输入值大则在次位增加重量 2^{n-1}。

(iii) 实施步骤(ii)的方法是将应该减去重量的位清 0(再将次位置 1)，应该增加重量的位置 1。将该数值再送入比较器进行比较，且这种比较一直进行到最低位的比较结束为止[1)]。

1) 原著此处使用的数据分别是 $2n$ 和 $(2n-1)$，根据原理，应更正为 2^n 和 2^{n-1}。——译者注

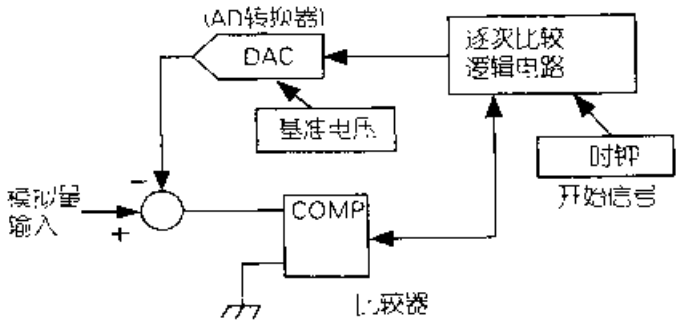


图6.6 逐次变换式AD转换器[10]

以上比较的结果即是AD转换值。由于直到比较全部结束后比较器才能得出结果，因此，必须附加采样-保持电路。此方法在一个通道时能够达到100kHz的采样频率(n个通道为$1/n$)，可以用于音频检测及机电控制等方面。

4. 串并联比较方式

此种方法将高位数据用DA转换器进行比较并可快速转换，将转换后的结果和信号的差进行AD转换，形成低位数据。这是一种精度高且速度快的转换方式。

5. 并联比较型AD转换器

这种转换器由若干梯形电阻和比较器的并联电路和译码器、输出闩以及时钟等构成，转换速度极快。

(1) 若转换比特数为n，基准电压用梯形电路($2n-1$)等分，再分别并联上比较器。

(2) 每个时钟周期可实现与$2n$个输入数据的比较，同时高速计算出转换值，再将结果用译码器转换为数字量，用闩电路保存下来。

并联比较型不需要采样-保持电路，但由于有多个比较器并行存在，电路负担较大。并联比较型可用于摄像机中以防止手持振动，或用于VTR的数据读取和图像处理以及数字示波器的宽带、高速检测等领域。通常，转换的数据位数为8位。

6. 其 它

一种较常用的产品是Analog Devices公司生产的8位AD转换器"AD7575"，利用5V单一电源供电，转换时间为5μs，接口也较简单。

表6.1对AD转换器的转换方式进行了比较。

表 6.1 AD转换器的比较

方式	类型	变换时间	分辨率	用途和特点
积分型	双积分型	10ms～1s	12～16位	温度及一般检测用
	Δ-Σ型			直流或交流，精度高
比较型	逐次比较型	1μs～1ms	8～12位	振动、声音及一般检测用中速、中精度、必须有采样-保持电路
	并联比较型	10ns～1μs	4～8位	摄像机解析、数字示波器用高速、电路规模大

6.2.4 分辨率和精度

8位的AD转换器可以产生 $2^8 = 256$ 种标记值，称为8位的分辨率。可见，分辨率是指AD转换器能够将相邻的值细分到什么程度[31]。例如，既有 -128～+127℃测量范围而刻度为1℃的温度计，也有0～25.5℃刻度为0.1℃的温度计，这些数据确定了满量程值和分辨率。

AD转换器的精度是指变换后的数字量的准确程度，其间的误差是非线性的，包括偏移误差和增益误差等多种误差。精度与分辨率是衡量AD转换器的两个不同指标。表6.2显示了AD转换器的比特数和精度。

表 6.2 AD转换器的位(bit)数和精度

位数	分辨率	精度/%	动态范围/dB
8	1/256	0.39	49.9
10	1/1024	0.10	62.0
12	1/4096	0.024	74.0
16	1/65536	0.0015	98.1
20	1/1048576	0.0001	122.2

例题 6.3 若以0.01%的精度进行温度检测，试问应该选择什么样的AD转换器？

❖解答❖ 由于温度变化反应缓慢，使用积分型16位的AD转换器比较适合。

6.2.5 AD转换用语[33]

◆ 分辨率：也称分解度，是指AD转换器可以分辨的最小模拟量，即如表6.2所示的 $1/2^n$ 形式的值。

◆ 精度：如表6.2所示的用%表示的分辨率，此外还包含线性误差和偏移误差等。

◆ 偏移误差:模拟量输入为0时,数字量输出不为0的基准点的误差与满量程的百分比。用偏移误差调节器可以调整偏移误差。

◆ 增益误差:为了使数字量输出达到满量程的输入电压误差。可以在偏移误差调整后,再调整增益误差。

◆ 线性误差:输入和输出的线性程度表示为满量程的百分比。

◆ 量化误差:参照6.1.2节。

◆ 转换速度:从开始执行转换指令到数字量输出为止的时间。

◆ 收集时间:从保持模式切换到采样模式时,采样-保持电路的输出新数据的时间。

◆ 缝隙时间:从保持模式切换到采样模式时,采样-保持电路与输入信号电路完全脱离的时间[33]。

◆ 设置范围时间:从输入信号变化开始到输出信号变化为止的延迟时间。若为DA转换器,则指从数字量输入开始到将模拟量输出调整到LSB的±1/2倍范围为止的时间。

例题 6.4 说明分辨率与精度的不同。

❖解答❖ 略(参考前述内容)。

6.2.6 输入输出电压类型

◆ 单极型:处理仅为正或负的数据,如0～+10V等。

◆ 双极型:处理正、负两种极性的数据,如-5～+5V等。

上述方式可以通过跳线进行设置,也有一些控制板可以利用软件设置。

◆ 标准二进制码:12位(4096刻度)AD转换器的分辨率为0.00024414×FS(基准电压)。对于单极型的0～+5V及FS-5V情况[31],有:

二进制　0对应　0V

4095对应　(+5-0.0012207)V(即+5-0.00024414×FS)

因为+5V已超过了满量程故不能测定。

◆ 偏移二进制码:同样为12位双极型(图6.7)-5～+5V的FS,有:

二进制　0对应　-5V

2048对应　0V

4095对应　(+5-0.0024414)V(即+5-0.00024414×FS)

这是通常使用的方式。

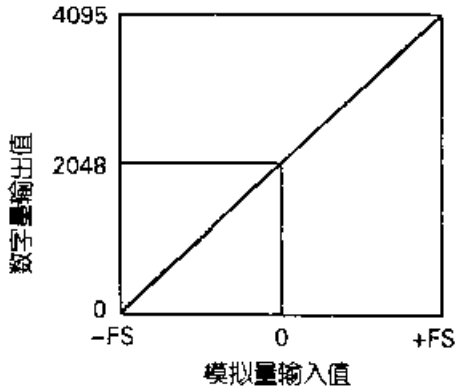


图 6.7 12 位 AD 转换器(双极型、偏移二进制)

- 补充形式二进制码:将数值量的 0 和 1 单纯地反转而来的码。
- BCD 码:直接显示 AD 转换值时使用的码。

6.2.7 数据传输

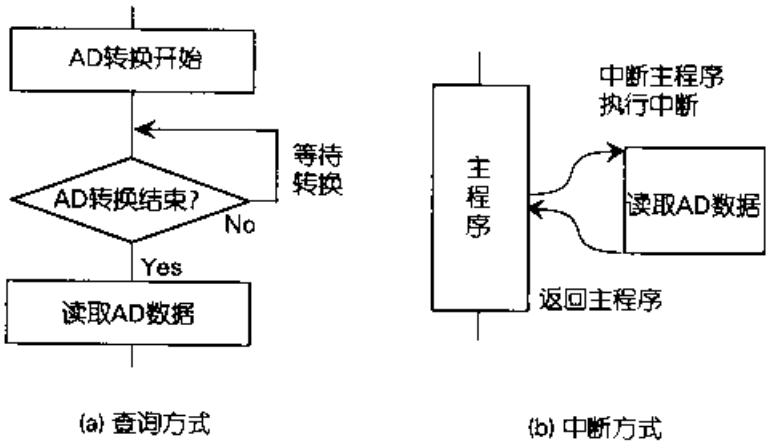
1. *查询方式*

在程序中等待 AD 转换结束标志(称其为**标志检测**),若标志位被置 1 则读取一个转换数据。图 6.8(a)显示了此过程:

(I) AD 转换开始

(Ii) 检测到转换结束

(iii) 读取转换的数据



(a) 查询方式　　(b) 中断方式

图 6.8 AD 转换的顺序

利用查询方式时,程序必须等待 AD 转换结束才能继续下去。对于传输时

间在 100μs 以上的 BASIC 等语言中，不可能进行更快的采样。但在 C 语言中，由于传输时间只为其 1/10 即 10μs，故可以采用 50kHz(20μs)左右的采样频率。

2. 中断方式

如图 6.8(b)所示，程序一直处于正常运行状态，AD 转换按如下方式处理：

(1) 如果 AD 转换结束，则用定时器中断引发使 CPU 执行中断。

(2) 临时终止正在执行的程序，执行中断例程。

(3) 读取转换数据，传送到内存后，再返回到原来的程序。

由于每 n 个通道可以转换一个采样数据，故此方法适合于机械控制。但对于现在的计算机来说，这种数据传输方式还是不够快的。

3. DMA 传输方式

以前所介绍的所有传输方式都需要借助 CPU 的寄存器，由 CPU 直接管理，这无疑增加了 CPU 的负担。DMA(Direct Memory Access)传输方式[31]则不然，虽然数据传输指令由 CPU 执行，但对总线的控制移交给了 DMA 控制器 IC，不须 CPU 的介入即可与外部设备直接通讯，因为整个过程不需要通过软件实现，使数据处理速度得到了很大提高。当使用 DMA 控制器 8237A(Intel 公司)进行传输控制时，CPU 释放总线控制权，由 DMA 代替 CPU 操作内存并执行传输指令。不过，由于目前计算机本身的串行传输速度越来越快，这种 DMA 方式的优势也越来越不明显。

4. 总线主控器

作为 DMA 的替代品，PCI 总线的总线主控器方式已成为目前数据传输方式的主流。SCSI 及图像控制卡等都使用快速的总线主控器方式。

5. FIFO 方式

此为 First-in/First-out 的缩写，工作基础是对采样数据的自动缓存。即 FIFO 用 4k～8kB 的专用存储器存储 AD 输出数据，计算机端设有将旧数据按顺序读出的机构(管道)。一般地，如果 FIFO 存储器中的数据已满一半(用 FIFO-Half 信号检测)则发出向 CPU 传送数据的指令。即使在数据的传输过程中，来自 AD 转换的新数据也可继续存储到 FIFO 存储器，并利用 FIFO-Half 信号将旧数据逐次向 CPU 高速传送。当然，也可以将数据逐个进行传送。

例题 6.5 说明查询方式和中断方式的区别。

解答 略(参考前述说明)。

6.3 AD 转换的实施

6.3.1 AD 转换程序

在机器人技术中，从传感器读取一个信号，依此进行判断，再输出一个控制信号是经常执行的动作。或者，也可能是一次从传感器读入多个信号，用模糊或神经元等方法进行判别，产生一个输出信号。对于 AD 板来说，即使采用新的 FIFO 方式也需要逐个读取数据进行处理，其基本用途也不同。

这里所讨论的结构是用定时器周期性对每个数据执行 AD 转换的控制板，可如下考虑选择条件：

(1) 若在系统中有其它的 DA 板或译码器板使用定时器，可利用同一定时器进行 AD 转换，用查询方式记录一个 AD 转换结果。

(2) 在以 AD 转换为主导的控制中，可以使用定时器中断方式进行时间控制。

Interface 公司的 IBX-3148(ISA 总线型)是满足上述条件的产品之一。该板的输入电压不是 ± 5V 而是 ± 10V，属于 12 位的 AD 转换。此转换器采用 8 个通道的差动输入，8 通道同时工作时转换时间为 10μs，最高采样频率 100kHz，使用 8259 中断(转换结束中断和定时器中断)。

6.3.2 查询方式

这里我们使用 DOS 平台，利用查询方式从通道 CH1 到 CHn 读取 AD 转换数据。目前，由于软件商直接提供了各种功能函数包，这使软件设计变得相对容易，因为用户只需要将函数进行适当地"组合"即可以形成自己的程序[34]，不必都从底层做起。以下是 Interface 公司提供的工作于 DOS 平台的程序。

```
//dosadl.c : AD-board IBX-3148, differential 8-chan, bipolar 5V, 10 V
//conv time : 10 us, 8-chan simultaneous
//interrupt : 8259, timer interrupt, timer: 8254, 8 MHz, sampling freq : 20 kHz
//Compiler : Microsoft C ver. 6.0 / Microsoft C/C++ ver 7.0, large mode
//          >cl /AL xxx.c s3118 c0.c
//Copyright (C) 1995 Interface Corp.
# include <stdio.h>
# include <dos.h>
```

```
# define ADR          0x0320          // AD板的I/O端口地址
# define NNchan 2                     // 读取AD转换数据的通道数
# define MMdata 1024                  // sampling data/channel
typedef unsigned int DATA;            // 读取AD转换的通道数
int Chn ;
//以下函数在该公司提供的库s3118C0.C中定义
//可以下载
extern void ad_ch (int) ;             // 切换通道CH
extern void ad_st(void) ;             // AD转换开始
extren DATA ad_data (void) ;          // 读取AD转换数据
extern void ad_ctrl(int,int,int, int); // AD板的动作控制
extern void ad_rng(int, int);         // 设置AD板的输入范围和输入模式
extern void wait_busy(void) ;         // 等待AD转换结束
void main (void)
{
    int ch ;
    DATA dat;
    Chn = NNchan ;
    ad_rng(12, 1) ;                   // 设置AD板的输入范围和输入模式
                                      // 输入范围+-5V,差动(differential)输入
ad_ctrl (0, 0, 0, 0);                 // 无定时器和外部触发器
                                      // 禁止EXINT IN和EXTRG IN
    for(ch = 0; ch<Chn; ch++)
    {
        ad_ch(ch) ;                   // 切换到ch通道
        ad_st( );                     // AD转换开始
        wait_busy( );                 // 等待AD转换结束
        dat = ad_data( );             // 读取AD转换数据(12位)
    }
}
```

6.3.3 FIFO方式

此处所述为Micro Science公司的ADM-640 AT(ISA总线型),12位,可4通道同时采样,提供1kB的FIFO存储器,最高采样频率可达500kHz[35],支持2种FIFO标志检测。

(1) 用not-empty标志监视1个数据,以实现单个数据的传送

(2) 用not-half-full监视512个数据,向CPU按块传送。

此处所示为使用not-empty标志进行单个数据传送的AD转换。

```
// po1 640e2.c ; (Co) MICRO SCIENCE ADM-640 AT Sample program
outp(iobase + TRIG, SOFT_TRIG);          //用软件触发开始 AD 转换
for (j=0 ; j<mmdata ; j- + )             // mmdata:数据个数
{
    for(i = 0; i<chann; i+ + )           //通道个数
    {
        while(((stat = inp(iobase + STATUS))&.
            NOT_EMPTY) ! = 0x010)        //监视 FIFO 的状态
        data[i][j] = inpw(iobase);       // iobase:AD 转换基址
    }                                    //读出一个数据存入用户定义的缓冲区
}
outp(iobase + TRIG, STOP);               //采样结束
```

6.3.4 中断方式

该方式在定时器的每个时间段用中断方式转换 1 个数据，产品软件只有 Inetrface 公司的 IBX-3118，可以下载。依其功能进行适当设置或修改后，能够适应 IBX-3148 等高速控制板。因为篇幅较长此处从略。该产品可以说是 DOS/V 中断方式的样板，希望读者能够自行研究。

6.4 DA 转换器

6.4.1 DA 转换电路

DA 转换器(DAC:Digital to Analog Converter)由参考电压或电流、电阻网络和开关等组成，用于再现模拟信号及信号发生器等方面。转换方式有 T 型(梯形)和脉冲调幅型等多种，这里仅给出了最具代表性的 T 型电阻式 DA 转换器模型，参见图 6.9。T 型即为"梯形"之意，其特点是用值为 R 和 2R 的电阻组成。因为电流在 OP 放大器的负输入端进行累加，故称为电流累加型 DA 转换器。计算公式如下：

$$V_{out} = -V_{in}\left(\frac{bit3}{2} + \frac{bit2}{4} + \frac{bit1}{8} + \frac{bit0}{16}\right) \tag{6.3}$$

这种电路的优点是只要保证电阻值的相对精度，即可构成高精度的 AD 转换器。此外还有电压累加型DA转换器，它将T型电阻连接在OP放大器的非

反相端。实际使用中，通常使用电压输出型的DA转换器。

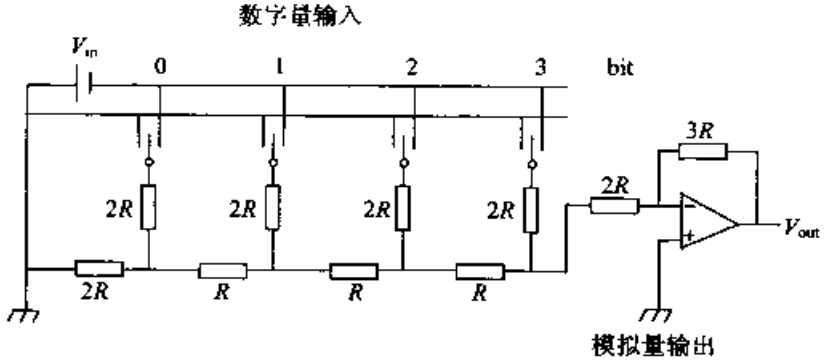


图6.9 T型DA转换器

6.4.2 DA转换器的实现

在接收传感器输入信号的同时控制电机时，AD端和DA端都会产生采样信号。这里我们使用Contec公司的DA板DA12-4(ISA总线型)[36]。该板为12位、4通道，转换时间为5μs/通道，并配有可编程定时器(interval timer)8254。该产品有完整的DOS/V平台函数库，可以在每个时钟脉冲进行DA转换，并可在此期间读取DA数据。

这里，我们参考Interface公司的IBX-3118的示例程序(属公开程序，可下载)编制了DA转换器用的定时器中断头文件。事实上，大多数制造商并不会公开这些软件，因此我们认为，给出这些头文件对读者是很有帮助的。

1. 板的配置

图6.10为Contec公司的DA接口板示例。使用时需要进行板上的地址

```
bit          D7      D6      D5      D4      D3      D2      D1      D0
(output)
ADR+ 0   |-------------------------------||-------||-------------|
              da data(lower)                 not-use     channel      // chan: 0~3
         1   |-------------------------------------------------------|
                                da data(higher)
         2   |-------|                                       |-------|
             all reset                                      trig reset  // trigger set: 1
         3   |-------||-------||-------||-------||-------||-------------|
             digital   out     trig    timer     x       interrupt level   // IRQ: 3~7
               out     mode    mode    gate              set               // timer start: 0x10
         8   |-------------------------------------------------------|     // timer stop:  0
                                counter #0 data                           // internal trigger: 0
         a   |-------------------------------------------------------|     // single out mode:  0
                                counter #1 data                           // x: any
         c   |-------------------------------------------------------|
                                counter #2 data
         e   |-------          -------------------------------------|
                                counter CWR
(input)
ADR+ 1   |-------||-------||-------||-------||-----------------------|
             BDC'ing   ext     x       trig-in           x
                       trig            status
```

图6.10 DA板的示例(Contec公司:DA12-4)

设置(调节板上的地址设置柄),如 300h。然后,在程序中定义相应的宏:

```
#define ADR 0x300
```

这样就可以在程序中正确使用板地址了。

2. 通道数据

```
d≪4; d = d || channei;
                                    //设置(ADR+0)地址和(ADR+1)地址的数据
outp(ADR+0, lower-data);
outp(ADR+1, higher-data);
```

3. 设置触发器

```
outp(ADR+2, 0x01);   //触发时启动定时器
```

4. 有关中断定时器的设置

定时器时钟为 4MHz,用 CLK#0(除数 c0)、CLK#1(同 c1)和 CLK#2(同 c2)分频得到需要的频率,其中的除数用于频率的除法计算,即:

(接线)4MHz→CLK#0→CLK#1→CLK#2→CLK OUT

(除数)　　c0　　c1　　c2

除数的设置条件:通常应满足 c0<c1<c2,c0 * c1 * c2 应大于 2,同时小于 0xffff。

定时器模式:使用模式 2。

例如,对于间隔为 1ms 的设置,有 4000k/2/2/1000,即 c0=2,c1=2,c2=1000,分频后为 1kHz。

```
outp(ADR + e, 0x34) ;          //clk # 0 CW
outp(ADR + 8, 0x2) ;           //lower data
outp(ADR + 8, 0) ;             //upper data
outp(ADR + e, 0x74) ;          //clk #1 CW
outp(ADR + a, 0x2) ;           //lower data
outp(ADR + a, 0) ;             //upper data
outp(ADR + e, 0xb4) ;          //clk #2 CW
outp(ADR + c, 0xe8) ;          //lower data
outp(ADR + c, 0x03) ;          //upper data
outp(ADR + 2, 0x1) ;           //trigger set
```

5. 8259 中断设置

```
(i) unsigned int irqtab[ ]= {      //中断向量号表
        0x00,0x00,0x00,0x0b,
        0x0c,0x0d.0x0e,0x0f,
        0x00,0x00,0x72,0x73,
```

```
        0x74,0x00,0x76,0x77 };
(ii) unsigned int msktab[j] = {          //中断屏蔽位表
        0x00,0x00,0x00,0x08,
        0x10,0x20,0x40,0x80
        0x00,0x00,0x04,0x08,
        0x10.0x00,0x40,0x80};
(iii) 中断使能方法
    IRQ5: d=(inp(0x21) & 0xdf) ;   // IRQ5:用 - mask 解除屏蔽
          outp(0x21,d) ;
    IRQ9: d=(inp(0x21) & 0xfb);    // slave connecting bit - mask
          outp(0x21,d);
          d2=(inp(0xa1) & 0xfd) ;  // IRQ 9 : - mask
          outp(0xa1,d2) ;
(iv) 中断结束处理:
    IRQ3 : outp (0x20,0x20);       // Master PIC 的一般 EOI
    对于 Slave PIC:
          outp(0xa0,0x20);         // EOI to master
          _asm
          {  mov    al, 0bh        ;read status
             out    a0h, al
             jmp    next           ;wait
          next :
            in      al, a0h        ;read ISR register
            and     al,ffh         ;check slave int
            jnz     end ;
          ;no other int
            mov al, 20h
            out 20h, al
    }
```

实际的程序可参考附录D。

6.4.3 自行设计的PC-9800系列用(C总线)DA板

此处所述为笔者自行设计的PC-9800系列用(C总线)DA板,文中给出了电路图、模型及输出控制程序。

图 6.11 显示了制作后的 DA 板的电路图(2 通道部分)。此控制板能够实现 12位的DA转换,可用于在机器人控制中向电机控制器输出控制指令等方

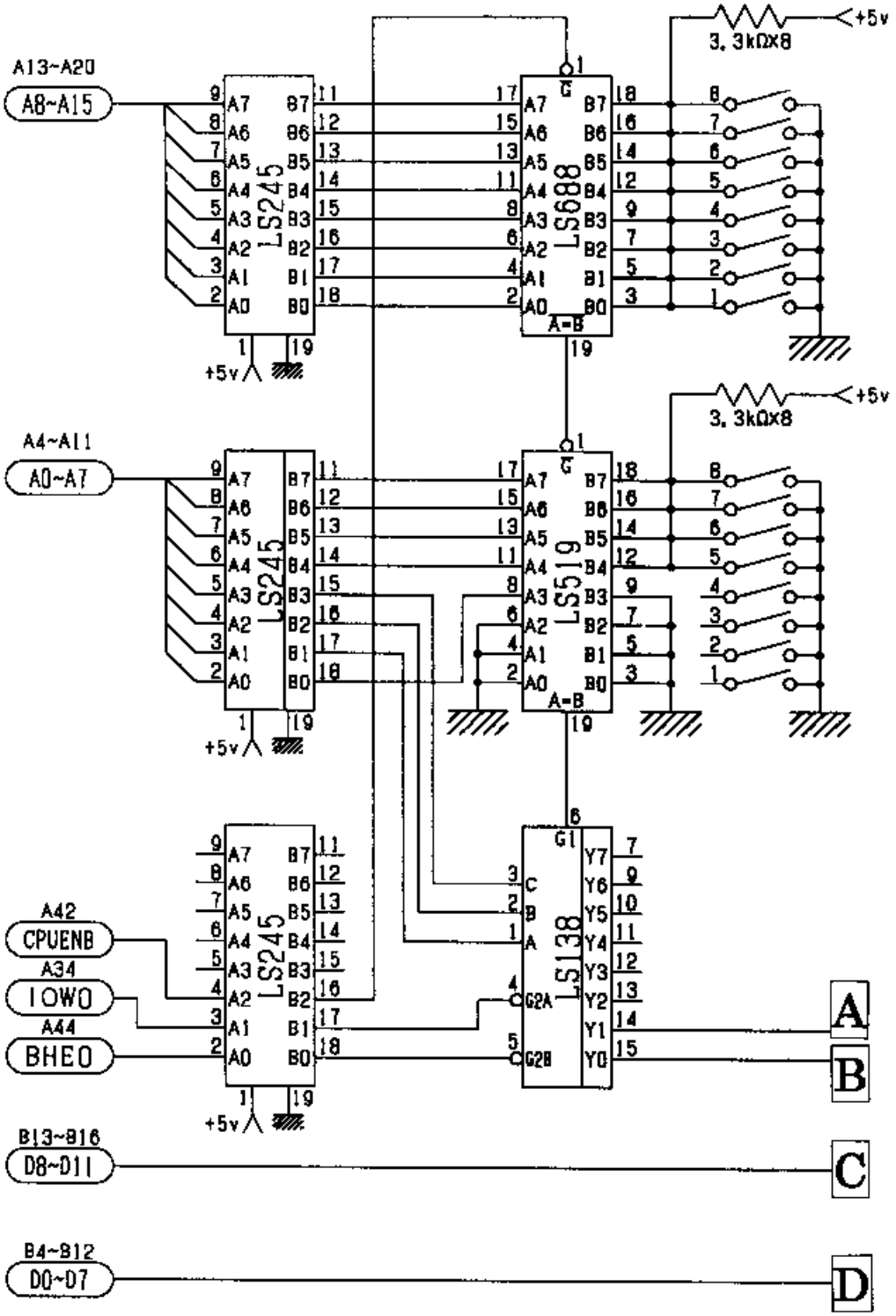


图 6.11(a) DA 板电路图(2 通道部分)

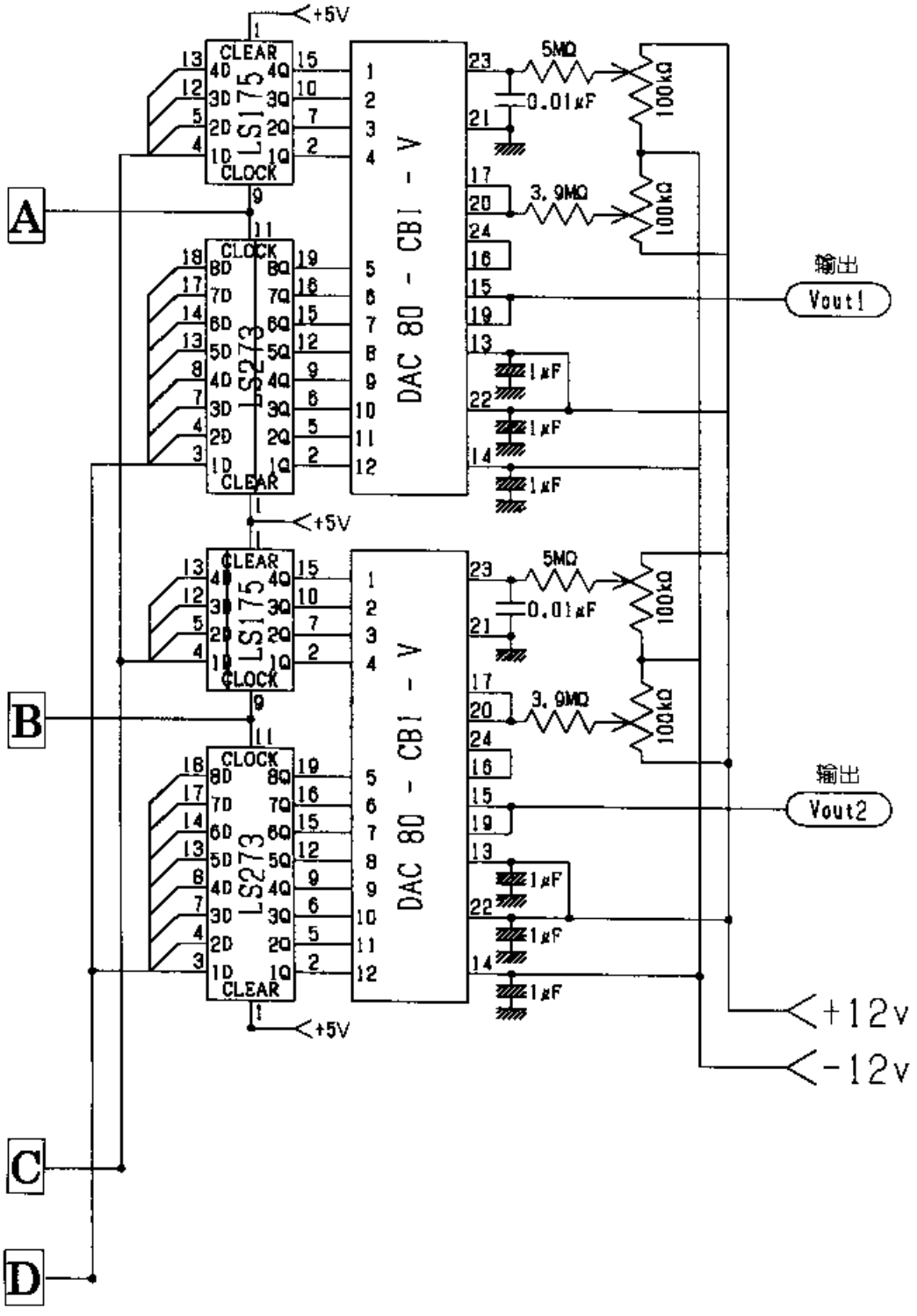


图 6.11(b) DA板电路图(2通道部分)

面。此控制板主要由地址译码部分、选择部分和DA转换部分构成,以下将对各部分给以简要说明。

1. 地址译码器(74LS688、74LS519)

在地址译码部分,根据前述对计数板的说明,将来自地址总线的输入数据与板上设置的I/O地址相比较,以此来判断是否有输入给该板的数据。输入给该板的数据模式如图6.12所示。其中的高位12位用于I/O地址的识别,低位第2～4位供选择部分使用,LSB恒为0。

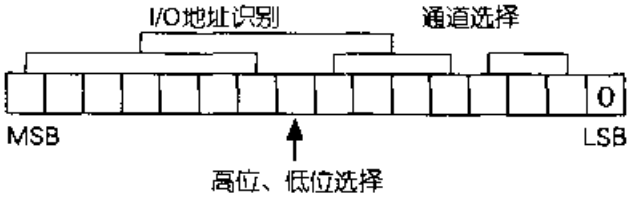


图6.12 输入到DA板的数据

2. 选择部分(74LS138)

在该板中必须进行与前述的计数板相同的通道选择。此选择部分的功能是在CPU发出OUT命令时变成活动状态,并用输入数据低位的2～4位选择通道。

3. DA转换器(DAC80)

在该板中,各通道都配置一个12位的DA转换器DAC80,并设置与12位的输入数据相对应的－10～＋10V输出(双极型输出),即与输入0x000(0)对应的是9.9951V的输出,与输入0xFFF($2^{12}-1=4095$)对应的输出为－10V,与输入0x8FF($2^{11}-1=2047$)对应的输出是0V。

该DA板的输出控制程序如下,使用的编译器是Turbo C++ ver.4.0。

```
#define DA_CH1 0x03D0                    // DA板1通道的I/O地址
void main( )
{
    int da_data;
    da_data = (int)...;                  // da_data=0～4095
    outport(DA_CH1, da_data);            // 输出到DA板
}
```

将以上程序与3.5节中的计数板(编码板)的输入控制程序相结合,即可以驱动DC电机等执行装置并读取旋转角度,从而实现机器人控制等任务。在第9章中,给出了一个利用这些接口板控制1轴机器臂的方法和程序示例。

练习6

问题 6.1 简述数字化的优缺点。

问题 6.2 用传感器实现精度为0.1%的检测、控制运算，应该使用多少位的AD转换器？

问题 6.3 简要解释下述名词：

(1)查询方式　　(2)偏移二进制码　　(3)总线主控器方式

问题 6.4 在关于控制板的定时器设置中，假定以base address为基址，地址0x08为定时器控制字，地址0x0a为clk#0，地址0x0c为clk#1，地址0x0e为clk#2。clk#0的out端与clk#2相连，用clk#0和clk#2实现采样分频。板的基本频率为8MHz，试制作20kHz的采样频率。假定定时器使用模式3。

问题 6.5 有4位的DA转换器，假定模拟电压的范围为±4V，请给出数字量输入和模拟量输出的对应关系图。

问题 6.6 使用12位的双极型、偏移二进制的DA转换器(FS为±5V)，生成峰值为±2.5V，1000个数据组成周期为1 s的循环锯齿波(参考图6.13)。

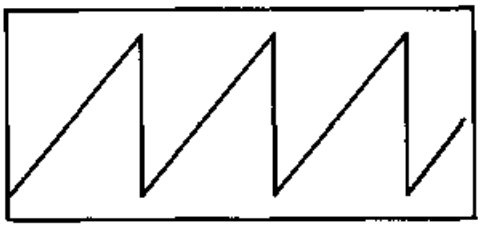


图6.13　锯齿波

chapter 7

BIOS和DOS

操作系统OS（Operating System）是计算机中最基本的软件。用户使用的所有软件都需要通过操作系统才能执行。基于磁盘的操作系统称为DOS，此外还有支持多任务的OS-2、面向通信的UNIX和Windows等多种操作系统。自从Windows95和NT登场，仅通过点击鼠标和菜单来完成与外部通信、文字处理和图形制作等工作已成为可能，此时用户甚至可以不必关心硬件的存在。但是，这种方式不是面向检测及控制接口的，检测控制领域仍然依赖C和DOS平台进行信号处理。因此，检测控制工作者必须学习C和DOS，以及更底层的BIOS等接口。

7.1 微机的层次结构和 BIOS

7.1.1 BIOS

BIOS(Basic Input Output System)是位于软件最底层的常驻设备驱动程序,是为了操纵计算机中安装的基本输入输出设备(如键盘、字符 CRT、图形 CRT、磁盘、RS-232C 和时钟等)而设置的最基本的程序模块。BIOS 固化在计算机的 ROM 中。

计算机启动时,先从特定的地址开始执行,称为**自举**(bootstrap)。启动时的自举、内存检测、外部设备检测、时钟和工作中的滴答声都来自 BIOS。这些程序存储在 CMOS 的 **SRAM**(Static Random Access Memory)中,用备份电源(锂电池)供电。MS-DOS 即由这部分 BIOS 程序启动,并接管系统工作。同样,Windows 也是在 BIOS 启动后运行的。

7.1.2 8086 系列的层次结构

Intel 公司的 16 位 CPU 总称为 8086,Pentium 家族属于该系列中更高档次的兼容机。在 Intel 系列的 CPU 中,16 位的 BIOS 系统被称为"实模式"系统,采用 DOS 模式(单任务)工作。以 Windows 为代表的 32 位 OS 被称为"保护模式"的系统,以多任务方式工作。

1. 软件的层次

DOS 作为最先执行的应用程序被放置在图 7.1 所示的高端位置,以便能够存取各种 BIOS。这意味着,在开发 DOS 平台的检测控制软件时,了解必要的有关 BIOS 的知识是必须的。即使是开发最新流行的 PCI 总线接口板的存取程序,也需要对 BIOS 知识有足够的了解。

PC/AT 兼容机的 BIOS 有不同的种类,与机器有关,通常是互相兼容的,但与有些软件可能产生冲突。

2. 系统开销

在 Windows 环境中,用户程序不能直接执行中断或对 I/O 进行存取,这些操作只能在 Windows 的统一管理下进行。这不仅使处理时间加长(称为系统开销),也使程序本身变得十分庞大。

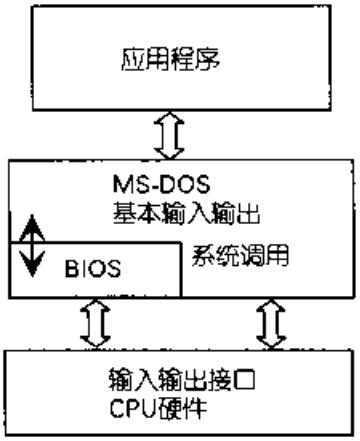


图 7.1 计算机软件[31]

7.1.3 Windows 环境中的 BIOS

Windows 中支持所谓的 **PNP 技术**，即插入 PCI 总线控制板后，系统可以马上产生相应的驱动程序。不过，目前机器所配备的标准外部设备越来越多，这只要按下述方式查看一下系统的配置即可明了：

在 Explorer 的“控制面板”中双击“系统”图标、再选择“设备管理器”页、用鼠标右键单击“计算机”、并在右键菜单中选择“属性”菜单项。

从属性窗口中可以发现，中断号 IRQ 通常都不是空的，即使是空的也有可能与某些设备相关，致使很多情况下对扩展板的中断号设置为空仍不能使其起动。众多的外部设备很容易导致 IRQ 冲突，致使设备不能正常工作。当然，我们不能仅仅因为安装一种设备时产生冲突就不使用该设备。不过，此时必须借助 BIOS 进行重新设置。

例如，有一块扩展板，可以使用 IRQ 的 10、11、12 和 15 中的任何一个中断请求实现数据处理，但安装时不管选择其中的哪一个都不能使该板正常工作。对于这种情况，可以删除目前不使用的 IRQ10(USB 端口)则该板正常工作，并释放对 IRQ3(通信端口 COM 2)的占用。此时，需要使用机器中的 Award BIOS 进行设置：

(i) 启动计算机，在运行 DOS 并出现“Press DEL to enter setup”提示信息时按 DEL 键，进入 BIOS 设置程序。

(ii) 选择“Chipset features setup”并按 Enter 键，再选择“Onboard Serial Port 2”(IRQ3 占用)，按 PageUp 键使之变成 disabled 状态。

(iii) 选择“PNP and PCI setup”，使表中的“USB IRQ”(IRQ 10 占用)项改成 disabled。

(iv) 上述操作使 IRQ3 和 IRQ10 项变成了"used by ISA"。现在,在 BIOS 中重新按下述顺序(操作顺序)指派与 PNP 对应的系统资源(system resource):

(非 PNP ISA 总线卡)→(PNP ISA 总线卡)→(PCI 总线卡)

这使主控制器 PIC 的 IRQ3 和从控制器 PIC 的 IRQ10 被分配在 ISA 扩展板上,系统资源中的 PCI 和 ISA 冲突就不会发生了。

即使如此,扩展板(不止限于 ISA,也包括 PCI)也有不能正常工作的时候,此时应考虑更换机器。

7.1.4 DOS 环境中的 BIOS

DOS/V 系统中可参照用户手册利用 BIOS 菜单完成 HDD 和 FDD 登录、BIOS 设置、I/O 设备的设置等工作。不过,此处所说的 BIOS 并不是仅指 Windows 环境中的设置操作,而是指实现如下的更多一些的控制功能。

1. 中断向量

DOS/V 和 PC-9800 系列中的中断都使用存储在 0h～FFh(FFh:10 进制 255)范围的中断向量表。内存的 0h～3FFh 范围是向量使用的**中断指针表**,在此表中存放着如前所述的中断处理程序的首地址。

表 7.1 是 DOS/V 的全部中断向量,其中的 INT10h～1Ah 属于 BIOS 范围。与 DOS/V 不同,PC-9800 系列中的中断号 18h～1Ch 属于 BIOS 范围。通常,BIOS 被称为软件中断,而与键盘、CRT 等的附属硬件通讯所使用的中断称为硬件中断。

2. 系统调用

BIOS 之上是 DOS 等操作系统,这样的系统软件通过 BIOS 调用实现输入输出控制,称此为 DOS 的**系统调用**。

3. C 语言和 BIOS

C 语言是运行在 MS-DOS、DOS/V 平台上的应用程序,当然可以使用 BIOS。利用 C 语言中备有的相应库函数,用户可以很容易地调用 BIOS 实现输入和输出。

利用 C 语言进行 BIOS 调用有如下一些**优点**:

(1) 直接存取 BIOS(不通过 DOS),代码紧凑且速度快。

(2) 可以实现 DOS 系统调用中没有的功能。

(3) 可以使用 DOS 中没有的 GP-IB 等输入输出设备。

表 7.1 DOS/V 的中断向量[6]

软件中断	使用目的	内 容
INT 00h	CPU 使用	除法错误
INT 02h	同上	NMI 中断(不可屏蔽中断)
INT 05h	子例程	打印屏幕
INT 06h	同上	终止键
INT 08h	8259 控制器	定时器
以下至 INT 0Fh 范围是主 8259 的硬件中断		
INT 10h	BIOS	视频 BIOS
INT 11h	同上	取得设备结构信息
INT 14h	同上	串口:RS-232
INT 15h	同上	系统 BIOS
INT 16h	同上	键盘 BIOS
INT 17h	同上	打印 BIOS
INT 1Ah	同上	日历、时钟
INT 1Ch	句柄	定时器中断句柄
以下为系统约定(DOS 及 DOS/V 的系统调用)		
INT 20h	系统调用	终止程序
INT 21h	同上	MS-DOS 的功能调用
INT 22h	同上	程序终止时 DOS 返回地址(用户不能调用)
INT 23h	同上	^C(CTRL + C)或 Break
INT 33h	同上	鼠标
以下的 INT70～77h 范围是从 8255 的硬件中断		

注:PC-9800 系列中除 8259、BIOS 之外与此相同[22,26]。

7.1.5 关于寄存器

在使用软件中断时,必须事先了解寄存器的结构。CPU 8086 系列共有 14 个 16 位的寄存器。

1. 通用寄存器

指图 7.2 中的 AX 到 DX(也可以小写)的 16 位通用寄存器,它们都可以被分成一系列的 8 位寄存器如 AH、AL 等使用。这些寄存器用于通常的运算和逻辑判断。

2. 程序指针 IP

用于记录正在执行的程序中指令的段内地址(偏移地址)。

3. 索引寄存器 SI 和段寄存器 CS

这是为了使 16 位机器能够使用 20 位地址总线而设计的,用于表示地址。利用这些寄存器可以使系统寻址 1MB 的内存空间。

取指令地址:按程序计数器 = (IP) + 16 × (CS)公式计算。执行中断时将会修改 IP 和 CS 的内容。

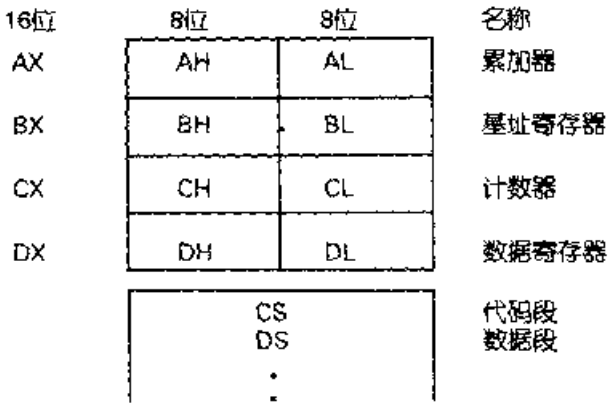


图 7.2 8086 系列通用寄存器[37]

4. 标志寄存器

执行软件和硬件中断时(前或后)要判断和干涉 CPU 的工作状态,图 7.3 说明了标志寄存器的内容。主要包括:

OF:溢出标志　　DS:方向标志　　IF:中断允许标志
TF:陷阱标志　　SF:符号标志　　ZF:零标志
AF:辅助进位标志　　PF:奇偶标志　　CF:进位标志

参照这些标志可以了解运算状态。例如,中断处理的执行结果会影响进位标志 CF,若成功将其置 0,失败则置为 1。加减法操作的借位和进位也影响 CF 标志。

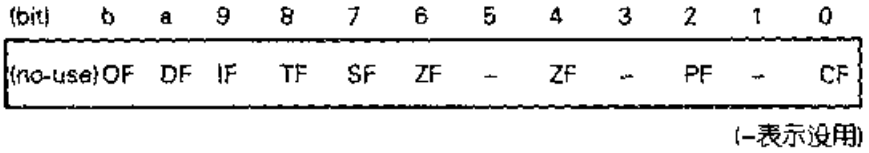


图 7.3 标志寄存器

5. 其它寄存器

其它的常用寄存器还包括 SP(堆栈指针)、CS(代码段)和 DS(数据段)等。

7.1.6 C 语言中寄存器的定义

1. 寄存器的定义

C 语言中应先用 # include <dos.h>进行声明,再定义**结构体和共用体**变量。

```
/* 字寄存器 */
struct WORDREGS
SREGS
{
  unsigned int ax;
  unsigned int bx;
  unsigned int cx;
  unsigned int dx;
  ...
  unsigned int flags;
};
```

```
/* 字节寄存器 */
struct BYTEREGS
{
  unsigned char al, ah;
  unsigned char bl, bh;
  unsigned char cl, ch;
  unsigned char dl, dh;
};
```

```
/* 段寄存器 */
struct SREGS
{
  unsigned int es;
  unsigned int cs;
  unsigned int ss;
  unsigned int ds;
};
```

2. union REGS inregs;及在 C 语言程序的定义

```
union    REGS                    /* 共用体 */
{
  struct    WORDREGS x;          /* 结构体 */
  struct    BYTEREGS h;
};
```

使用上述寄存器的共用体和结构体定义时,全部寄存器都用代表 int 类型(16 位)的 x 表示,而高位、低位分开时用代表 char 类型的 h(hex)表示。具体地说,在程序中使用寄存器时,它们是如下一些变量:

	(int 类型)	(char 类型)
通用寄存器:	inregs.x.ax	inregs.h.ah 或 inregs.h.al
基址寄存器:	inregs.x.bx	inregs.h.bh 或 inregs.h.bl
计 数 器:	inregs.x.cx	inregs.h.ch 或 inregs.h.cl
数据寄存器:	inregs.x.dx	inregs.h.dh 或 inregs.h.dl
代 码 段:	inregs.x.cs	
数 据 段:	inregs.x.ds	

3. 标志寄存器的定义

标志寄存器是用于了解和控制 CPU 的状态的 16 位寄存器,其内容如前所述。在 C 语言中可以参照上述的结构按如下方式使用标志寄存器:

标志寄存器:outregs.x.cflag

4. 软件中断

在 C 语言中,软件中断函数和寄存器的关系体现在如下的定义中:

```
union REGS                  /* 共用体 */
{
  struct WORDREGS x;  /* 结构体 */
  struct BYTEREGS h;
```

```
};
int intdos (union REGS *, union REGS *);
Int intdosx (union REGS *, union REGS *, struct SREGS *);
int int86 (int intno, union REGS *, union REGS *);
int int86x (int intno, union REGS *, union REGS *, struct SREGS
*);
```

其中：

(1) bdos、intdos和intdosx是**系统调用**0x21专用的中断。

例如，bdos可以按如下方式使用：

```
int bdos(dosah, dosdx, dosal);
```

参数dosah是ah寄存器的值，dosdx是dx寄存器的值，dosal是al寄存器的值。用准备好的各实参数值调用中断时，计算结果被填充到某些寄存器并结束函数调用。各寄存器值的含义可参阅DOS的程序员参考手册。

(2) 在函数int86( )和int86x( )中可以用于任何软件中断类型，参数intno为软件中断号。

(3) 函数int86( )用于不需要使用段寄存器的中断调用。

函数int86x( )用于必须使用段寄存器的中断调用。

具体用法可参考如下说明。

7.1.7 BIOS调用

1. C语言中的BIOS调用

在C语言中调用BIOS时，用户需要定义union REGS *、struct SREGS *类型的指针变量，且通常使用的是普通变量的地址。下述程序中，&inregs和&segregs是输入指针，&outregs是输出用指针。

```
#include <dos.h>
union REGS inregs, outregs;          /* 共用体定义 */
struct SREGS segregs;                /* 结构体定义 */
int intno;
……
int86x(intno, &inregs, &outregs, &segregs);
        /* 或者使用 int86(intno, &inregs, &outregs); */
        /* 从BIOS返回的值存储在outregs.x ax中 */
```

2. 定时器间隔的设置[24]

DOS/V：经过指定的时间后设置标志寄存器。

BIOS中断号：	int 15h
服务(功能)号：	ah=83h, al=00h
定时器设置：	cx:dx寄存器(输入)
结束设置：	es:bx

定时器复位：　　al=0
出错码输出：　　ah(输出)
标志设置：　　cf=0表示成功,1表示出错

这些值在下述程序中进行了设置,更详细的说明可参考书后的参考文献[24,28,29,37]。

```
#include <stdio.h>
#include <dos.h>
union REGS inregs, outregs;
struct SREGS segregs;
int set_time(unsigned int time0[])                    /* 定时器事件设置 */
{
    inregs.h.ah = 0x83;                               /* 服务号 */
    inregs.h.al = 0x00;                               /* 同上 */
    inregs.x.cx = time0[0];                           /* 定时器设置 */
    inregs.x.dx = time0[1];                           /* 定时器设置 */
    inregs.x.bx = time0[2];                           /* 微秒 1 */
    segregs.es = time0[3];                            /* 微秒 2 */
    int86x(0x15, &inregs, &outregs, &segregs);        /* BIOS 调用 */
    return outregs.x.cflag;
}
void main( )
{
    unsigned int timea[4];
        /* 使用 timea 是因为名字 timer 在 C 语言中已定义为其它用途 */
int err;
timea[0] = timea[1] = 0x0fff;                         /* 含义为 μs */
    timea[2] = timea[3] = 0x0000;                     /* 同上 */
    err = set_time(timea);
    if(err)……;
}
```

7.2 MS-DOS的系统调用

7.2.1 内部中断

DOS/V软件中断的功能包括表7.1的INT20h至27h范围。在约定的系

统调用中，指定中断号即可在软件中实现的中断调用称为内部中断。特别地，称 INT 21h 为功能调用。为了从 DOS 中执行 INT 21h 功能调用，需要用功能号填充 ah 寄存器。

1. 系统调用

在 C 语言中有系统调用 INT 21h 的库函数定义，函数原型为：

```
int bdos(int dos_func, unsigned dx_rej, unsigned al_rej);
```

表 7.2 系统调用 int bdos 的功能号参数 dos_func(部分)

dos_func 号	功 能
00h	程序终止
01h	键盘输入字符并回送
02h	向 CRT 输出字符
05h	向打印机输出字符
08h	键盘输入(无回送)
09h	向 CRT 输出字符串
0ah	输入字符串
0bh	检查键盘输入状态
0ch	清除键盘输入缓冲区
2ah	取日期
2ch	取时间

其中，

(1) 执行 dos_func 的功能时，需要用表 7.2 中的值填充 ah 寄存器。

(2) bdos 函数中直接给出 dos_func 的值，如：

```
bdos(0x2a, 0, 0);
    /* 读取日期值，与寄存器无关时填入 0 */
```

(3) dx_rej、al_rej 分别对应 dx 寄存器和 al 寄存器，若不需要可赋以 0。

(4) 结果存放在 al 寄存器中。

2. 使用寄存器共用体的系统调用

(1) intno 为中断向量号，即 21h 或 0x21。

(2) 参考 DOS 使用手册填充 * inregs 和 * outregs 的值。

(3) 语句 intdos(&inregs, &outregs); 与 int86(0x21, &inregs, &outregs); 功能相同。

3. 确认是否有键按下

下述代码通过系统调用测试是否有键按下。

```
#include <stdio.h>
#include <dos.h>
union REGS inregs, outregs;

unsigned char keyb_status(void)          /* 是否按键 */
{
```

```
    inregs.h.ah = 0x0b;                 /* 键盘状态 */
    intdos(&inregs, &outregs);          /* 若有键按下,outregs=0xff */
    return outregs.h.al;
}
unsigned char keyb_clear(void)          /* 清键盘缓冲区 */
{
    inregs.h.ah = 3;                    /* 按键后,清键盘缓冲区 */
    int86(0x16, &inregs, &outregs);     /* PC-9800 系列使用 0x18 */
    return outregs.h.al;
}

    unsigned char key_read(void)        /* 读字符 */
{
    inregs.h.ah = 0;                    /* 读输入的字符 */
    int86(0x16, &inregs, &outregs);
    return outregs.h.al;
}

void main( )
{
    unsigned char c0;
    c0 = keyb_status( );                /* 若有 1 个字符如'a'输入则 c0=0xff */
    if(c0)
{
    c0 = key_read( );                   /* 读字符'a'并舍弃 */
    c0 = keyb_clear( );                 /* 清键盘缓冲区使其变为 0x0 */
    }
}
```

假定程序开始运行时,键盘缓冲区为 0x0 并进行循环。若有键按下,函数 keyb_status( )的返回值为 0xff,即 outregs.h.al 的值。随后程序用 key_read( )函数读取输入的字符并舍弃,再用 keyb_clear( )函数清键盘缓冲区,使 outregs.h.al 又变为 0x0。根据此顺序,可以通过不使程序终止的键盘输入改变计算流程。若不执行 key_read( )函数,输入的字符在程序结束时仍会留在键盘缓冲区中。

7.3 DOS 操作

从 1980年开始至今已深为用户所熟悉的MS-DOS(MicrosoftDiskOp-

erating System)共发行到了 ver.6.2 为止,应用重点即转移到了 Windows 及 Visual C++。但就检测控制领域而言,从可靠性等方面考虑仍然使用 MS-DOS。当然,若不能直接从 DOS/V 用的纯 MS-DOS 入手,也可以使用 MS-VC (Microsoft Visual C++)的 DOS 功能。如果使用 IBM 系列机则可以从 PC-DOS 入手。在使用 DOS 时,内部和外部中断的使用十分频繁,这就要求对 DOS 有一定了解。

7.3.1 MS-DOS 的结构

1. 基本内存

因为 MS-DOS 本来是 16 位的 OS,因此只能存取 1MB 以下的内存。不过,因为图形 VRAM 占用高端的 380kB 内存,MS-DOS 自身使用的内存只能是低端的 640kB,用户的程序和数据也使用这部分内存。

2. 关于扩充内存

这里,为了扩充内存空间,引入了 UMB、HMA 和 DOS 扩展等诸多概念。

(1) UMB(Upper Memory Block)——位于图形 VRAM 的高端位置。用户可以使用 devicehigh 命令将设备驱动程序和缓冲区设置在此区域,以便少占或不占基本内存。利用 devicehigh 命令可以加载字符类型和 block 类型(制作 RAM 磁盘等)的设备驱动程序。

(2) HMA(High Memory Area)——位于 UMB 之上,大小为 64kB,DOS 可以将其组成 XMS 管理器使用。或者,也可以用其加载日文输入的 FEP 系统。

(3) EMS(Expanded Memory Area)——不属于 1MB 之内的附加内存区域。使用扩充内存时,DOS 需要在标准的 1MB 内存中设置特殊的"页框"(小内存块),并将扩充内存中的数据复制到页框内(或反之)才能使其被访问。

目前,1MB 以外的内存主要以 XMS(eXtended Memory Specification)方式安装和使用,它使用户可以使用更大的内存空间。

(4)EMM386.EXE 是实现 EMS 内存和 XMS 内存及 UMB 管理的驱动程序[38,39]。在 MS-DOS 中可以利用后续的 config.sys 文件进行如下定义:

DOS=HIGH 或 DOS=HIGH, UMB

在 Windows 普及之前,DOS 作为软件开发和应用的主流操作系统得到了充分的发展,为了运行应用程序,config.sys 文件和 autoexec.bat 文件扮演了重要的角色。Windows 中的 config.sys 和 autoexec.bat 是自动设置的,因此,尽管用户没有意识到它们,系统也可以正常工作。但在 DOS 平台进行应用程序配置时却经常需要检查和修改这二个文件。

DOS 模式由以下 3 个部分组成[40]:

io.sys、msdos.sys 和 command.com

应用程序处于它们之上。

3. command.com

从键盘输入 dir(列目录)和 copy 等命令时,由该程序进行解释并送给 msdos.sys。

4. msdos.sys

此程序的主要功能是对文件、内存和设备进行管理。

(1) 文件管理——打开文件和数据读写。

(2) 内存管理——分配和释放内存。

(3) 设备管理——磁盘存取、屏幕显示及字符输出等。

5. io.sys

此程序从 msdos.sys 接受命令,负责输入和输出,即执行磁盘(A:、B:等)、键盘、显示(CON)和打印(PRN)等的标准设备和低层的输入输出,不直接与外部程序交互。

6. config.sys

用户利用此文件设置各种设备的使用方法,称其为系统配置文件。

(1) device=驱动程序文件名;此命令将设备添加到系统中。

(2) adddrv/deldrv;启动后增加/取消设备驱动程序(仅字符型设备有效)。

在 DOS/V 机中,利用 SYSTEM Commander 等工具安装 2 个操作系统时,config.sys 可以按原来的 DOS 风格描述[38],并可以用 DOS ver.6.2 启动到驱动器 C:。

以下是在 DOS/V 机中加载 DOS 驱动程序的 config.sys 文件示例。很多字符型设备驱动程序可以通过 devicehigh 命令加载到内存高端,以减少基本内存的占用。改变 config.sys 的内容后,必须重新启动计算机。

```
C:\type config.sys                    ;DOS/V 环境
buffers = 30                          ;设置缓冲区数目,以提高文件操作速度
files = 30                            ;可并发打开的文件数目
DEVICE=C:\DOS\HIMEM.SYS
                                      ;high memory area(XMS)管理程序
DEVICE=C:\DOS\EMM386.EXE
                                      ;扩充和扩展内存管理程序
DEVICEhigh=C:\DOS\BILING.SYS
                                      ;使系统可以使用日语、英语软件
DEVICEhigh=C:\DOS\JFONT.SYS /P=C:\DOS\
                                      ;提供全角、半角字体
DEVICEhigh=C:\DOS\JDISP.SYS
                                      ;显示全角字符的驱动程序
```

```
DEVICEhigh = C:\DOS\JKEYB.SYS /106 C:\DOS\JKEYBRD.SYS
                        ;设定英语的 101 键盘和日语的 106 键盘
DEVICEhigh = C:\DOS\KKCFUNC.SYS
                        ;控制 MS-DOS 的日语假名汉字系统(FEP)的驱动程序
DOS = HIGH, UMB         ;可以使用 HMA、UMB
DEVICEhigh = C:\DOS\ANSI.SYS
                        ;屏幕、键盘的 Esc 序列控制等
DEVICEhigh = C:\DOS\MSIMEK.SYS /AI
                        ;加载日语 IME 输入法
DEVICEhigh = C:\DOS\MSIME.SYS /D * C:\DOS\MSIMER.DIC
                        /D C:\DOS\MSIME.DIC /C1 /N A1
                        ;C1:Shift JIS,A1:使用 EMS 内存,N:echo-line
DEVICEhigh = C:\DOS\PRINT.SYS /F:打印机驱动程序
@echo on                ;用此命令使后续命令在屏幕上显示,@表示本身不显示
shell = c:\command.com c:\ /E:1280 /p
                        ;shell = 命令处理器 command 的名字、
                        ;地点,/p 表示不使 command.com 中止
                        ; /E 设置环境变量区域的大小(单位为字节)
                        ; /P 表示启动时运行 autoexec.bat
rem DEVICE = b:\DOS\RSDRV.SYS
                ;RS-232 驱动程序。rem 表示此行为注释
rem DEVICE = a:\DOS\MOUSE.SYS
                ;鼠标驱动程序
```

译者注:根据 DOS 手册中的语法,上述命令中的 /p 和 /P 应该是同一个 /P,此开关使系统重新加载一个命令解释器的拷贝,因此不能用 EXIT 命令中止此命令解释器。

7. autoexec.bat

这是在计算机启动时能够自动执行的**批处理文件**。此文件通常用于设置环境变量和确定各种路径等。其中的**路径**是指系统对外部命令和文件执行搜索的地点。

```
b:\>type autoexec.bat
    C:\DOS\SMARTDRV.EXE /X
                        ;对磁盘缓存,X 表示取消缓存
C:\DOS\NLSFUNC.EXE C:\DOS\CONTRY.SYS
                        ;国别信息,用于 JP 和 US 的切换
C:\DOS\CHEV.COM JP      ;日语模式
PROMPT $p$g             ;将提示符从 A>的形式修改为 A:\>形式
@ECHO ON                ;显示以下命令
set orgpath = %PATH%    ;保存以前的路径
PATH d:\mifes55;d:\vz;e:\msvc\bin;d:\tc4\bin;
C:\DOS;d:\mozart;d:\;c:\
```

```
path %path%;d:\msinput\mouse
                                ;追加鼠标驱动程序路径
@echo off                       ;停止命令显示
SET TEMP=d:\                    ;指定工作目录
c:nis`                          ;执行名为 nisi 的 EXE 文件
```

7.3.2 文件系统

磁盘中的每个文件都有其属性，在 Windows 中可以通过鼠标右击查看文件的属性，而在 DOS 中可借助 VZ 编辑器[1]等查看文件的属性。

不同的属性对用户操作此文件进行了不同程度的限制：

R:read only,H:hidden,S:system,A:Archive,可修改或复制

(1) read only file——只读文件(不可修改或删除)。

(2) hidden file——隐含文件，不能修改或删除。但用 attrib(属性)命令修改其属性后可删除。

(3) xx.sys——DOS 的系统文件

(4) xx.exe——可执行文件

(5) xx.com——可以常驻内存的不超过 64kB 大小的可执行文件。用 exe2bin.exe 可以将.exe 文件转换成.com 文件。

7.3.3 层次目录

磁盘等存储设备中的文件存储空间通常被划分成如图 7.4 所示的树状结构，最顶层的根称为"根目录"，其它的树枝部分称为子目录。任何一个目录自身可以用"."表示，其父目录可用".."表示。例如，执行存放在 AB 目录中的可执行文件 xx.exe 时应输入如下的命令：

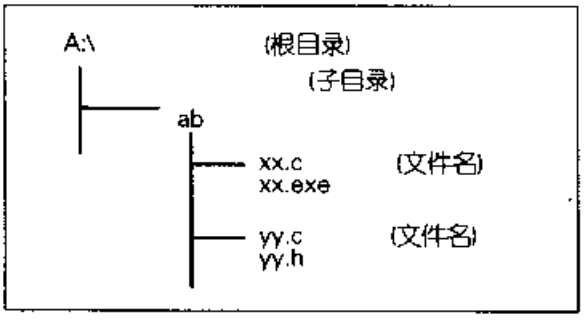


图 7.4 目录结构

1) VZ 在国内较罕见，可使用 DOS 本身提供的 attrib 命令。 ——译者注

A:\>cd ab

A:\AB>xx

若返回根目录可输入 A:\AB>cd . . 命令。

7.3.4 DOS 命令选粹

DOS/V 机中从 Windows 切换到 DOS 提示符或 DOS 模式后，即可以使用通常的 DOS 命令。事实上，即使到了 Windows 时代，文件操作也是至关重要的。初学者最好能逐个验证一下这些命令。

C:\>D:——切换到 D 驱动器

C:\>dir——列文件目录

C:\>dir f * ——显示文件名首字符是 f 的文件列表

C:\>copy c:\b01.c d:\ab\ * . * ——将 C 盘根目录下的文件 b01.c 复制到 D 盘的目录 ab 中，* . * 表示文件名字不改变

C:\>copy d:\b01.c prn——打印文件

C:\>del d:\ab\xx.c——删除一个文件

C:\BC>del * . * ——删除目录 BC 中的所有文件

C:\>md d:\ac——在 D 盘根目录下创建子目录 ac

C:\>rd d:\ab——删除子目录 ab(需要用 del 命令先删除该目录下的所有文件)

C:\>cd bx——change directory:切换到 C:\BX 子目录

C:\BX>cd by——再进入下级子目录 C:\BX\BY

C:\BX\BY>cd. .——返回到上一级目录 C:\BX

C:\AB>ren xx.c xy.c——将子目录 ab 下的文件 xx.c 改名为 xy.c

C:\>type autoexec.bat——显示 autoexec.bat 文件的内容

C:\>format a: /s——使用参数 /s 将 A:盘(软盘)格式化成系统盘

C:\>sys a:——传送 DOS 系统(MSDOS.SYS、IO.SYS)到 A 盘

C:\DE>yyy——执行 C 盘 DE 目录内的可执行文件 yyy.exe

下述命令体现了 DOS 的过滤功能：

C:\>dir | sort——用过滤功能将文件按顺序排列后显示

C:\>cl xxx.c | more——编译程序并将其错误信息分页显示

7.3.5 批处理

在执行一系列固定的DOS命令时，可以将这些命令组合成一个**批处理文**

件，为后续执行提供方便。批处理文件的扩展名必须为.bat。

```
ECHO ON                          ;允许屏幕显示正在执行的命令
PAUSE                            ;暂停
GOTO mm                          ;跳转到标号 mm 处
IF %1 = "x"                      ;若输入的字符是 x...
REM                              ;注释
```

1. 批处理文件例 1

此批处理文件名为 abc.bat，执行时应输入"abc 参数 1 参数 2"。

```
echo off
if "%2" = = " " goto ERR         ;若参数 2 为空则出错
if NOT EXIST %1 goto ERR         ;若参数 1 表示的文件不存在则出错
copy %1 %2                       ;文件复制
echo copied!
del %1
goto END
:ERR
echo ERROR
:END
```

2. 批处理文件例 2

此文件的功能是：输入"use xx"时，进入 xx 子目录，执行 autoexec.bat。在 autoexec.bat 文件中，记录了运行应用程序的环境设置方法和运行命令，最后，仅用"use xx"即可运行应用程序。此批处理文件 use.bat 的内容如下：

```
B:\
CD %1
AUTOEXEC.BAT
```

实际执行时可输入"use msc"。通过此例形式的批处理文件，可以将子目录 MSC 的环境设置用 autoexec.bat 实现，这样就可以用一个命令来整理编译 C 语言程序的环境。

7.4 标准通信接口

在计算机与外部设备的通信中，有**串行通信**和**并行通信** 2 种方式。向打印机传送数据采用并行方式，由于传输遵循打印机的数据传输格式，打印机不必向计算机返回数据。作为串行通信的代表是 RS-232C，控制简单但速度较慢。

在检测控制中，为了使外部设备和计算机之间能够在远距离达到快速通信的目的，经常使用 GP-IB[31]。进一步讲，计算机串行端口 RS-232C 是向用户开放的，最近在办公室中已用 LAN(局域网)实现了计算机之间的通信，而在家庭中通过电话线(Modem，调制解调器)作为中介使我们实现了在计算机之间的通信。

即使在机器人控制中，使用串行口 RS-232C 进行计算机和控制设备之间进行数据存取的场合也很多。我们需要了解在什么情况下使用串行接口(SIO)，什么情况下使用并行接口(PIO)。

7.4.1 RS-232C

串行接口 RS-232C 是由 ATT 开发的接口，它符合美国电子工业协会 EIA 标准，是用于连接调制解调器(数字电路终端设备：与计算机串行通信)的标准接口。MS-DOS 支持此标准，并将其作为异步传输的标准输入输出设备。RS-232C 接口主要用于：

(1) 利用公用电话线进行数字通信(通过 Modem)

(2) 计算机和外部设备的接口

(3) LAN

1. 电 缆

利用 RS-232C 的计算机外部设备包括 XY 绘图仪、数字化仪、图像扫描仪、测试仪和小型机器人等。为了实现与这些设备的连接，需要使用专用的 RS-232C 电缆。此电缆包括双绞线和同轴电缆 2 种，购买时应留意。通常使用双绞线电缆。

2. RS-232C 的驱动

在 MS-DOS 环境中：

(i) 在 config.sys 中加入命令 device=rddrv.sys

(ii) 使用初始化命令 speed 进行参数设置(注意先弄清楚存储器开关的内容)：

① 传输速率(波特率)——1200 bps～56kbps

② 数据位长——7 位或者 8 位

③ 奇偶校验——奇数或偶数

④ 停止位长——设置为 1 位或 2 位等等。

7.4.2 DOS/V 的 RS-232C

在控制程序中，视频和音频设备需要进行巨大数据量的运算，在将计算结果用于机器人控制，或将控制程序加载到小型移动机器人等场合，需要实现异种机之间的通信。例如，在笔者的研究中，一部计算机用于判别声音的方向，另一部计算机则需要根据此计算机所计算出的角度控制机器人的动作，二者之间需进行数据通信。此时，为了实现主计算机和外部设备中的子计算机之间的数据交换需要使用 RS-232C，并且也需要我们自己编制相应的接口程序。

1. 串行端口

DOS/V 机(AT 机兼容)有 2 个串口，通常 COM1 被 PS/2 鼠标器占用，COM2 用于串行通讯(此 LSI 与异步专用的 NS16550 相当)。

2. 通信软件

DOS/V 机的串行通讯 BIOS 的软件中断为 INT 14h，它比直接使用硬件中断的速度要慢些，且为了使程序简单，经常以 9600 bps 速率进行通信。在近期的高速通信中，使用了中断控制器 8259A 的硬件中断来代替软件中断。

3. BIOS 通讯

这里给出的是一个使用 COM2 端口和 INT 14h 中断，参数为 9600 bps、8 位、无奇偶校验，只传送一个字符的 RS-232C 程序。

```
// RS-232C 头文件:DOS/V RS-232C。参考书后的参考文献[24,28]
// :使用双绞线
// 初始化:ah 寄存器 = 0,port = COM1
bit7-5 (baurate)bit4 - 3 (parity)bit2 (stop)bit 1 - 0(word length)
100   1200bps   00    none    0   bit      00 reserved
101   2400      01    odd     1   bit      01 reserved
110   4800      10    none                 10 7 bit
111   9600      11    even                 11 8 bit
//发送:ah 寄存器 = 1
    (i) 参考状态设置 CTS、DRS 和 CD
    (ii) 若发送数据缓冲区的 bit13 为空则允许发送一个字符
    (iii) 检测是否处于发送中。出错时设置 al 寄存器的 bit7。
    // 接收:ah 寄存器 = 2
    (i) 读取 8250UART 中的数据
    (ii) 检察状态数据的 data ready 信号可判断是否有要读取的数据。
    // 状态:ah 寄存器 = 3
    (Register)    (status)
    ah: bit 7        time-out error
        6            发送用 shift register 的状态(1:empty)
        5            发送用 buffer register 的状态(1:empty)
        4            接收 break 信号
        3            错误帧
        2            parity error
```

```
            1           overrun error
            0           data-ready(接收数据存在)
    ah : bit7           CD (carrier detect)为 ON
            6           RI (bell) ON(CI)
            5           DSR (data set ready) ON
            4           DTS (clear to send) ON
            1           CD(carrier detect)变化
            2           RI(bell)的变化 (CI)
            1           DSK (data set ready)的变化
            0           CTS (clear to send)的变化
/* ……RS-232C BIOS…… */
//rs232c. h
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
union REGS inregs, outregs ;
int init_rs232c (int port, int param)          // initialize RS-232C
{     inregs.h.ah = 0 ;
      inregs.x.dx = port;
      inregs.h.al = param ;
      int86(0x14, &inregs, &outregs) ;
      return (outregs.x.ax) ;                  // al: modem status ; ah: kalsen status
}
int send_rs232c (int port, char char0)         // send RS-232C
{   inregs.h.al = 1 ;
      inregs.x.dx = port;
      inregs.h.al = char0;                     // send char
      int86(0x14, &inregs, &outregs) ;
      return outregs.h.ah ;                    // send status
}
int receive_rs232c(int port, char * char0)  // receive RS-232C
{
      inregs.h.ah = 2;
      inregs.x.dx = port ;
      inregs.h.al = *char0 ;                   // send char
      int86(0x14, &inregs, &outregs) ;
      *char0 = outregs.h.al ;
      return outregs.h.ah;                     // receive status
}
      int status_rs232c(int port)              // status RS-232C
{
      inregs.h.ah = 3 ;
```

```
    inregs.x.dx = port;
    int86(0x14, &inregs, &outregs) ;
    return outregs.x.ax;
}
```

7.4.3 停止模式

从停止状态出发进行数据发送有 2 种方法。

1. 全双工方式

来自键盘输入的数据，直接从 RS-232C 口输出，此时不产生屏幕显示。而来自 RS-232C 的编码直接显示。由此可见，RS-232C 的输入和输出是互相独立进行的。

2. 半双工方式

键盘输入直接显示到屏幕，同时保存到缓冲区，RS-232C 不产生输出。接收到 CR 码后数据才从缓冲区传给 RS-232C。从 RS-232C 接收的数据也在缓冲区暂存，待键盘输入结束后才显示该数据。

在将上述过程转化为程序时，需要增加一个时刻监视键盘输入和来自 RS-232C 的输入的例程。

3. 字符行的区分

为了将两行字符分隔开，文本文件中保存了如下的内容：

在 C 语言中："字符行 + \n"，此处的\n = 0x0a

在 MS-DOS 中："字符行 + CR/LF"，此处 CR = 0x0b，LF = 0x0a

不过，若在 C 语言打开一个文本文件并用 RS-232C 传送，系统会自动对字符行进行调整。

练习 7

问题 7.1 DOS/V 中光标的位置用 video BIOS 的 INT 10h 设置，调用时，寄存器 ah 的值为 02h，bh 为 0，dh 为行数，dl 为列数。试编写内嵌汇编语句的 C 语言程序，将光标定位在 5 行 10 列。

问题 7.2 编写程序，用 MS-DOS 的系统调用向字符设备输出一个字符，已知功能号如下（DOS/V 与 PC-9800 系列都相同）：

ah 寄存器 = 02h：输出到 CRT，(Ctrl + C)有效

= 04h:辅助输出(串行设备输出)
= 05h:输出到打印机,(Ctrl + C)有效
= 06h:输出到 CRT

dl 寄存器 = 应该输出的字符代码

问题 7.3 假定现在的提示符是 B 盘的根目录,使用 MS-DOS 命令将 b:\bx 内的文件 bcd.doc 复制到 c:\cx 内的文件 cde.doc。

问题 7.4 编制一个清除键盘缓冲区的程序。

问题 7.5 使用 RS-232C 的头文件编制一个数据通讯的主程序。这里,要求使用 8 位、9600bps 进行数据传输,无奇偶校验,1 位停止位。

DOS/V的显示器可以工作在VGA（Vidio Graphics Array）640×480点阵和XGA的1024×768点阵等高分辨率模式，均可由软件设置。事实上，显示处理作为视觉接口是非常重要的部分。

8.1 视频模式

8.1.1 DOS/V 机的显示方式

1. VGA

DOS/V 机的视频 LSI 是具有 IMB-PC 的标准图形规格的 VGA(Video Graphic Array),对应的显示状态是:

VGA——分辨率:640×480 点,颜色数:16 色

高于 VGA 分辨率的规格是 Super VGA,还有提高图形处理能力的所谓"图形加速卡"。显示卡共有以下 3 种视频模式:

(1) CGA(Color Graphic Adater)文本模式:支持字符前景色、背景色的设置;

(2) 扩充 CGA 文本模式:网格;

(3) 图形模式。

2. 模式切换

在 DOS/V 机中,显示卡上有 128kB 的显示用 VRAM,显示图形或字符时需要使用显示 BIOS。文本画面和图形画面的模式可以切换。在画面上显示图形和文本时,需要分别加以控制,单独进行。这与 PC-9800 系列机不同,DOS/V 不支持文本和图形同时工作(即要么使用文本方式,要么使用图形方式)。利用视频 BIOS 使用软件中断的 INT 10h 到 INT 1Ah 实现输入输出控制,不经过 DOS。C 语言中也备有调用 BIOS 的函数。为了实现上述功能,需要加载 $ disp.sys[1] 驱动程序。

8.1.2 Esc 序列

1. Esc 序列字符

Esc 序列字符(特殊字符)并不是由 DOS/V 本身处理的字符,而是驱动程序 ansi.sys 所必须的。ansi.sys 可以控制显示模式的变化、光标移动、文字颜色和画面背景的设置等等[28]。

Esc 序列字符是指[15]:

1) 原文如此,但英文 DOS ver 6.2.2 中此程序应为 display.sys。——译者注

ESC 字符——"\x1b" + "["

利用 WZ 编辑器可以编辑控制字符。本书中用表 8.1 中的 C 语言语句输出 Esc 字符序列,形式为:

printf("\x1b""[xxx]");

注意字符串中不能包含空格。为了能够通过编译,应该将字符串分成 2 部分来写。

表 8.1 DOS/V 的 ESC 序列

ESC 序列字符	意 义
printf("\x1b""[2h");	80×25 点阵单色字符模式,英语
printf("\x1b""[3h");	80×25 点阵单色字符模式,日语
printf("\x1b""[2J");	清屏
printf("\x1b""[xxm")	字符颜色:xx 为附录 B 中的颜色号

8.1.3 颜色模式

在 DOS 平台中,使用支持 VGA(Video Graphic Array)标准的 16 色文本方式以及 640×480 点阵图形模式的模拟显示器。如果使用 Windows 平台,采用加强分辨率时可以达到从 1024×768(1670 万色)到 1600×1200(65000 色)的图形显示能力。

(1) 模式 03h——英文 80×25 点阵彩色字符模式(即文本模式),支持日文显示,此为图形方式的缺省模式

(2) 模式 12h——英文 640×480 点阵 16 色图形 VGA, 80 列×25 行

(3) 模式 72h——日文 640×480 点阵 16 色图形模式, 80 列×25 行,相当于_VRES16EXCOLOR 模式

(4) 模式 73h——日文 80 列×25 行点阵彩色字符仿真扩展 CGA 模式,可返回到原来的 72h 模式。

其它模式可参照附录 E。

8.1.4 视频 BIOS

以下给出的是使用 INT 10h 进行视频模式设置的程序示例[24,29]:

1. 视频模式设置

```
#include <stdio.h>
#include <dos.h>
```

```
#include <conio.h>
union REGS regs;
void set_videomode(void)                    /* 视频模式设置 */
{
  regs.h.ah = 0;                            /* ah为服务号 */
  regs.h.al = 0x12;                         /* 彩色图形和文本模式 */
  int86(0x10, &regs,&regs);
}
```

2. 视频信息

```
void get_videomode(void)
{
  regs.h.ah = 0x0f;
  int86(0x10, &regs, &regs);
  videomode = regs.h.al;
  activepage = regs.h.bh;
}
```

3. 光标定位

```
void locate(int x, int y, int page)         /* 光标定位 */
{
  regs.h.ah = 2;
  regs.h.dl = (unsigned char)x;             /* x坐标 */
  regs.h.dh = (unsigned char)y;             /* y坐标 */
  regs.h.bh = (unsigned char)page;
  int86(0x10, &regs, &regs);
}
```

4. 读取光标位置

```
void get_locate(int * x, int * y, int page)
{
  regs.h.ah = 3;
  regs.h.bh = page;
  int86(0x10, &regs, &regs);
  *x = regs.h.dl;
  *y = regs.h.dh;
}
```

8.2 图 形

8.2.1 用 C 语言控制 DOS/V 的图形

在 VGA 图形模式下,屏幕上共有 640 × 480 个点。在 MS-VC + + 的 C:\MSVC\include\graph.h 中定义了如下的图形模式:

```
short _far _cdecl _setvideomode(short);
/* arguments for _setvideomode( ) */
#define _MAXRESMODE          (-3)
            /* graphics mode with highest resolution */
#define _MAXCOLORMODE        (-2)
            /* graphics mode with most colors */
#define _DEFAULTMODE         (-1)
            /* restore screen to original mode */
#define _VRES2COLOR          17      /* 640×480,2 color */
#define _VRES16COLOR         18      /* 640×480,16 color */
#define _VRES16EXCOLOR       0x72    /* 640×480,16 color,(80
                                        ×25) */
#define _CGAEMEXTEXT         0x73    /* 80×25 text, 16 color */
```

视频模式的设置函数_setvideomode( )的参数,即是指以上 8 种模式设定值,其中使用频繁的视频模式包括如下几种:

(1) _VRES16EXCOLOR——640 × 480 点阵,16 种颜色图形和 80 × 25 点阵字符。

(2) _MAXCOLORMODE——选择最适当的模式。通常选择的即是_VRES16EXCOLOR 模式。

(3) _DEFAULTMODE——返回设置前的状态。

在使用 MS-VC for DOS 及 MS-C(Microsoft C ver. 6)进行图形程序设计时,要用 include 命令包含 graph.h 头文件。graphics.lib 库文件在 MS-VC 的 setup 时已事先与编辑模式结合,以后在编译程序时系统会自动连接该图形库。

(4) 使用函数_setvideomode( )进行图形模式初始化时,物理画面的左上角为原点(0,0)而右下角为(639,479)。

(5) 使用函数_setviewport( )在物理画面内设置视口,使得图形只显示在视口内。

(6) 使用函数_setwindow( )在视口内设置要输出图形的范围。例如,左边第一个参数值为1,第3象限左下角可为(-1000., -1000.),第1象限右上角可为(3000., 3000.)等等。

(7) 使用函数_setlinestyle( )设置线型,若实线则为0xffff,点线可为0xc0c0等。

以下是使用MS-VC for DOS及MS-C编写的图形程序示例。

```
//MS VC或MS C的图形
//display:VGA color 640×480,Japanese 80×25 text
//compile:d:\>cl xxx.c graphics.lib
#include <graph.h>                                  /* 图形头文件 */
int style, color;
void main( )
{
  _setvideomode(_VRES16EXCOLOR);                    /* 图形模式初始化 */
  _setviewport(0, 0, 639, 479);                     /* 设置物理屏幕 */
  _clearscreen(_GCLEARSCREEN);                      /* 清屏 */
  KEISAN( );                                        /* 执行某些计算 */
  _setwindow(1, -1000., 1000, 2000., 3000.);        /* 图形描述范围 */
  style = 1;
  if(style == 0) _setlinestyle(0xffff);             /* 实线 */
  if(style >= 1) _setlinestyle(0xc0c0);             /* 点线 */
  _setcolor(color);                                 /* 颜色 */
  display( );                                       /* 绘图函数 */
  _settextposition(y, x);                           /* 指定坐标 */
  _settextcolor(7);                                 /* 字符颜色7为白色 */
  printf("\a");                                     /* 响铃 */
  _clearscreen(_GCLEARSCREEN);
  _setvideomode(_DEFAULTMODE);                      /* 结束 */
}
```

在附录E中给出了包含图形字体操作的示例程序。

8.2.2 TC++的图形

在使用c:\tcc xxx.cpp graphics.lib编译程序时与MS-VC不同:

```
#include <graphics.h>
initgraph(&gd, &gm, "c:\tc4\bgi");
closegraph();
```

图形函数的名字中都不包括前缀"_"。

8.2.3 图形的拷贝

即便我们已经用 MS-C 将一幅图形显示到了屏幕上，但要将这幅图形在 DOS/V 中用打印机打印出来还会遇到一些问题。为此需要采取如下的措施：

(1) c:\dos\graphics：执行 DOS 的 graphics.com 命令并使其常驻内存

(2) 将 C 语言计算的图形数据显示在屏幕上

(3) 将打印机设置为 ESC/P 打印模式，按 printScreen 进行屏幕打印。

虽然此图形命令可用于日文和英文模式下的打印，但在 IBM 和 HP 系列的模式下可能不正常。

(4) 当然，若输出图形时使用的是从 Windows 环境切换而来的 DOS 窗口，按 PrintScreen 键时图形将被复制到剪贴板而不是打印机。此时，可以启动 Windows 的"画图"等软件，将剪贴板中的图形粘贴到文件中，再将图形反色即可打印。

8.3 鼠标器接口

8.3.1 鼠标器

鼠标器是一种以坐标点作为输入输出的指针设备。如图 8.1 所示，鼠标器内部有 X 轴方向和 Y 轴方向的 2 个光旋转编码器，轴的转动所形成的脉冲被转换成 8 位的计数。鼠标器有 3 个点击开关(按键)，分别为 LEFT、MID 和 RIGHT，其中的 MID 键通常被用于控制画面滚动或开关等方面。对于 DOS/V 中检测控制程序，通常在截取波形、点击拾取峰值等操作时需要使用鼠标器。

(1) 移动距离——相对X轴向 θ 方向移动L (mm)距离时，发生的脉冲数为[42]：

$$P_x = kL\cos\theta \pm \text{误差}$$

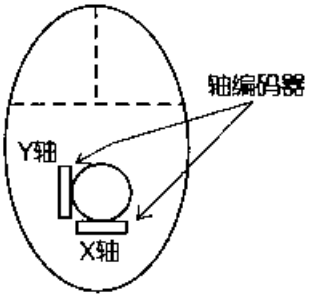
$$P_y = kL\sin\theta \pm \text{误差}$$

其中的$k\approx4$，误差在 15 个脉冲以内。采用与通常的编码相同的方式进行方向判别，处理图中(c)的$X_A - X_B$之间(及$Y_A - Y_B$之间)的脉冲的相位超前或滞后。

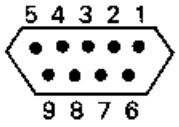
(2) 位置坐标的范围——当鼠标光标处于屏幕的左上角(0,0)到右下角(639,479)的范围内时，计算机可以检测到鼠标的位置。

(3) 米奇——鼠标移动距离的表示单位,1 米奇是 1/100 英寸(0.25mm)。

(4) 米奇/点比——用 1 点相当于多少米奇的比。在普通模式彩色显示器中,鼠标光标的大小是 16×16 点阵。



(a) 鼠标器



(b) 连接

引脚	
1	+5V
2	X_A
3	X_B
4	Y_A
5	Y_B
6	Left
7	Mid
8	Right
9	GND

(c) 引脚分配

图 8.1 鼠标器接口

(5) 中断类型——INT 33h(DOS/V 和 PC-8800 系列)

(6) 鼠标驱动程序——在 config.sys 文件中用 device = mouse.sys 或 mouse.com 命令加载鼠标器驱动程序。

8.3.2 鼠标驱动程序的组成

从 Windows 切换到 DOS 模式后,不能直接使用 PS/2 鼠标。为此,可以在 Windows 的系统菜单中选择"运行"菜单项,再指定 CD-ROM 的 DOS/V 用鼠标驱动程序文件,例如:

1. DOS/V 用的鼠标驱动程序

可以按下述方式打开鼠标器驱动程序所在的目录:

>Windrv\mouse\disk1\setup;选择 custom

>MS-DOS Tools and Controls;选择 MS-DOS 驱动程序

>restart

然后,执行鼠标驱动程序 mouse.exe:

C:>\mspoint\mouse\mouse.exe

其它机种的驱动程序位于别的目录中。

2. 路径设置

在 autoexec.bat 文件中添加(或修改)path 命令:

path %path%;c:\mspoint\mouse ;%path%表示包括原来的路径设置

添加上述命令后,就可以使用 PS/2 鼠标器了。

8.3.3 DOS/V 的鼠标

1. 模 式

PC-9800 系列中,允许字符模式和图形模式混合工作(同时可独立进行控制),DOS/V 则不允许二者混合工作,对于用户来说,必须编制适合文本方式和图形方式的 2 种鼠标操作程序。无论如何,画面的左上角总是(0,0),右下角为(xn,yn),此处的模式即是指前节中所述的模式。

2. 鼠标函数库

有很多公司提供使用 INT 33h 的鼠标操作函数库,甚至可以在文本和图形两种方式下通用[28,29]。在编制文本方式的鼠标函数时,可以参考书后的附录 F 中的列表。

例如,下述函数可用于读取鼠标光标的当前位置的X、Y 坐标:

```
#include <stdio.h>
#include <dos.h>
union REGS inregs, outregs;
void get_mouseXY(int * a, int * x, int * y)
{
  inregs.x.ax = 0x03;                    /* 读坐标和按键的状态 */
  int86(0x33, &inregs, &outregs);          /* 鼠标器系统调用 */
  *x = outregs.x.cx;
  *y = outregs.x.dx;
  *a = outregs.x.bx;              /* 按键状态:bit0 = 1(左键 ON) */
  return;                  /* bit1 = 1(右键 ON),bit2 = 1(中间键 ON) */
}
```

8.3.4 用鼠标端口实现脉冲计数[41]

因为鼠标端口使用 8255,若知道此端口地址,直接存取即可实现对来自编码器的脉冲等的计数。若使用 V-F 转换器作为中介,用鼠标器端口也可以进行 AD 转换。对 PC-9800 系列此问题是清楚的,可以用该端口实现计数功能,程序略。

练习 8

问题 8.1 开发图形程序时，常常发生机器"死锁"后屏幕上残留一些图形的情况，试编制一个输入 clrscr 即可以进行清屏且返回到 DOS 屏幕(文本模式)的程序并编译成 exe 文件。

问题 8.2 利用 DOS/V 的 BIOS 将视频设置为日文 640×480 点阵、16 色的彩色图形 VGA、80 列×25 行的视频模式。

问题 8.3 设置视频为 DOS/V 图形模式，再定义使用屏幕上半部分，左下角为(－50.，100.)而右上角为(＋50.，＋100.)的用户窗口。

问题 8.4 编写函数，其功能是判别鼠标器的左键还是右键被按下，并返回鼠标指针在屏幕上的坐标值。

问题 8.5 编制函数，控制鼠标器光标的显示与关闭。

chapter 9

执行装置接口

执行装置（actuator）是从电源得到能量并以此驱动其它设备动作的发生装置，如电机、线圈、液压和气压缸等等。较新的装置包括形状记忆合金、压电体、静电制动器和超音速电机等。在机器人、OA设备（复印机、打印机和传真机）及家用电器（VTR、CD）等的动作机构中，传动装置是必不可少的部分。

仅以动力驱动的设备而论，控制用的执行装置通常称为伺服执行机构。自然地，控制用的电机称为伺服电机，一般主要是指DC电机和AC电机，有时也包括步进电机和液、气压电机。这些电机要求对转速和转矩有较大的控制范围，在加减速和正反转方面具有耐久性，并且具有较高的反应速度。在控制领域中，计算机（微处理器）控制是目前应用的主流，而计算机与执行装置之间具有简单的接口是先决条件。至于计算机本身则从台式计算机开始到机顶式计算机乃至袖珍计算机等都有应用。

9.1 步进电机概述

9.1.1 特 点

步进电机依据电气脉冲信号运转，故也称之为脉冲电机。对应 1 个输入脉冲，电机将产生一个步长的转角，且电机在下一个脉冲到来之前会保持现在的位置，成为停止状态。这种依据输入脉冲数运转并保持状态的特点是 DC 电机所不具备的。

因为脉冲是数字信号，这恰是计算机所擅长处理的数据类型。从 20 世纪 80 年代开始开发出了专用的 IC 驱动电路，今天，在打印机、磁盘驱动器等的 OA 装置的位置控制中，步进电机都是不可缺少的组成部分之一。总体上说，步进电机有如下优点：

(1) 不需要反馈，控制简单。

(2) 停止时可保持转矩。

(3) 与微机的连接、速度控制(启动、停止和反转)及驱动电路设计比较简单。

(4) 没有角度累积误差。

(5) 没有换向器等机械部分，不需要保养，故造价较低。

但是，这种电机也有自身的缺点：

(6) 在体积、重量方面没有优势，能源利用效率低。

(7) 超过负载时会破坏同步，低速工作时会发生振动和噪音。

9.1.2 种 类

从转子和定子的结构可将步进电机分为 3 类[33,43]。结构图可参见专门书籍，此处略。

1. PM 型(Permanent Magnet 型)

转子采用多磁极的圆筒形的永磁钢，在其外侧配置齿状定子。用转子和定子之间的吸引和排斥力产生转动，转动 1 步的角度一般是 7.5°。这种电机的价格较低。

2. VR 型(Variable Reluctance 型)

采用高导磁材料构成齿状转子和定子，近来已很少使用。

3. HB 型(Hybrid 型)

这是 PM 和 VR 的复合型产品，其转子采用齿状的稀土永磁材料，定子则为

齿状的突起结构。此类电机的步进角很小(0.9°～3.6°),但具有很高的转矩,在计算机相关设备中多用此类电机。

9.1.3 特 性

步进电机的脉冲速率和发生脉冲之间具有如图 9.1 所示的关系。当步进电机的运转速度升高时转矩变小。由于在电机启动时会产生很大的转矩,因此必须在缓慢地启动后,逐渐加快速度。停止时恰好是其逆过程。与电机特性相关的术语包括如下:

最大静止转矩——静止时保持的最大转矩

最大启动转矩——无负载时用输入频率在 10 pps 以下的较低频率即可驱动电机时所发生的最大转矩

自启动范围——电机突然运转时能够与脉冲保持同步的范围

牵入转矩——在能够保持与输入脉冲完全同步的情况下启动的最大转矩

临界(牵出)转矩——与输入频率一一对应,且能够使负载也同步运转的极限转矩

通过范围——按通常方式牵入会有振动发生,缓慢地增加频率和负载时,能同步运转的范围称为通过范围。

最大自启动频率——电机无负载时能同步运转的最大频率

最大应答频率——超过自启动范围,脉冲频率缓慢升高时能够保持同步运转的最大频率

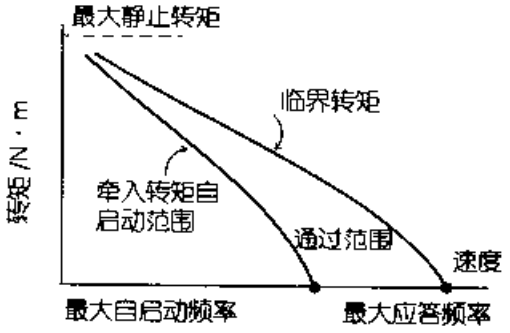


图 9.1 步进电机的驱动特性[26,27]

9.1.4 励磁方式

步进电机包括 2 相、4 相和 5 相电机。图 9.2 中给出了采用 8255 的输出驱

动步进电机的示意图。在 4 相电机中有 4 组线圈，若电流按顺序通过线圈则使电机产生转动。2 相电机中有 2 组线圈。从图 9.3 中可以发现，在各线圈中引出中间端子，因此，若以中间端子为基准即可实现 4 相，称这 4 相为 A、$\overline{A}$、B 和 $\overline{B}$ 的励磁相。通常使用的电机都是 4 相电机，而励磁方式中有 1 相(单向)励磁、2 相(双向)励磁和 1-2 相(单－双向)励磁方式。此外，如果转动的方向不正确，可以交替 1、2 号端子或者 3、4 号端子。

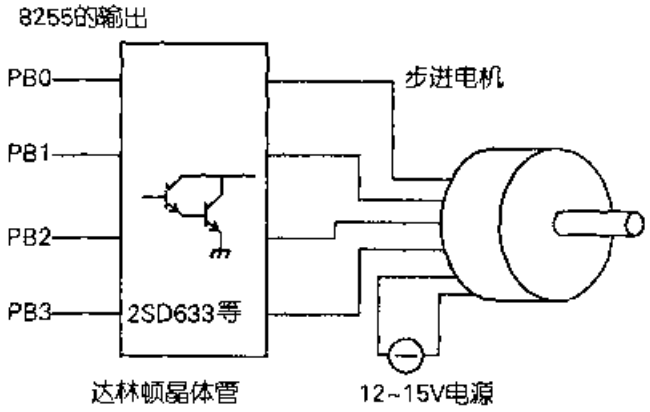


图 9.2 用 8255 的输出驱动步进电机[11]

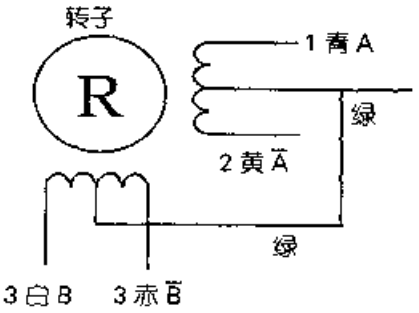


图 9.3 4 相步进电机的端子

1. 1 相励磁方式

如图 9.4(b)所示，按 A、B、$\overline{A}$和$\overline{B}$的顺序总是仅有 1 个励磁相有电流通过，因此，对应 1 个脉冲信号电机只会转动 1 步，这使电机只能产生很小的转矩并会产生振动，故很少使用。

2. 2 相励磁方式

如图 9.4(b)所示，按 AB、B $\overline{A}$和$\overline{A}\overline{B}$等的顺序总是只有 2 相励磁，通过的电流是 1 相励磁时通过电流的 2 倍，转矩也是 1 相励磁的 2 倍。此时，电机的振动较小且应答频率升高，目前仍广泛使用此种方式。

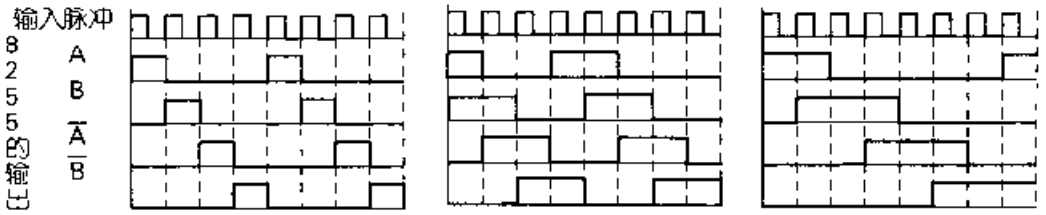


图 9.4 励磁的时序[30,33]

3. 1-2 相励磁方式

如图 9.4(c)所示，按 1 个，2 个，……的顺序交替进行线圈的励磁驱动。与前述的 2 个线圈励磁方式相比，电机的转速是原来的 1/2，应答频率范围变为原来的 2 倍。转子以滑动方式转动。

驱动电路的输入脉冲序列的安排方式决定了以上 3 种励磁方式，市场上有专用的 IC 出售。

9.1.5 驱动装置的外围设备

为了实现对步进电机的驱动，必须附加相应的辅助设备，包括生成指令脉冲的控制电路、供给电力的电源以及驱动线圈励磁的驱动电路等。

1. 脉冲发生器

可以采用专用的脉冲发生器，也可以利用计算机软件产生脉冲的方法生成指令脉冲。

2. 直流电源

必须配有符合电机的额定电压、电流为 6～24V，0～5A 左右的直流电源。

3. 驱动电路

此为励磁相控制部分，用于产生对应励磁模式的励磁时序，读取方向信号 CW/CCW(顺时针/逆时针)等数据以切换转动方向。通常步进电机和对应的驱动电路一起出售，可以直接使用。例如，CoParu 公司的小型步进电机 SP4 中就设置有驱动电路 DM-402X，仅需要来自外部的时钟脉冲和控制切换转动方向的输入，即可进行矩形波整形、脉冲分配、CW/CCW 切换(转动方向切换)等。每一相的输出电流是 170mA。

9.1.6 串联电阻的作用

电机线圈、电阻 R 和电感 L 组成串联电路。高速运转(频率升高)时，因为 L 的阻抗增大，致使通过的电流和转矩变小。为了改善这种情况，需要在如图

9.3 所示的绿色端子上串联电阻并接地。同时，还需要在电源和线圈之间加入串联电阻。

(i) 外部电阻是线圈电阻(约 70Ω)的 2～6 倍。

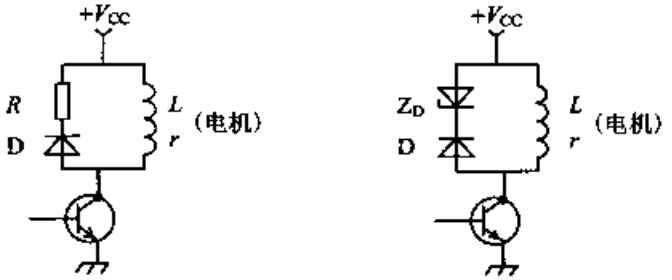
(ii) 这使阻抗和时间常数变小，能够加快应答速度。

9.1.7 尖峰电压问题

因为步进电机具有感应负载，电机启动和停止时的电流变化会导致很大的反向感应电势，很容易烧坏晶体管。为了抑制这种电涌电流，可以使用如下方法(参见图 9.5)：

(1) 将串联的二极管和电阻一起并联到电机中(图中(a))。因为串联了电阻 R，图中(a)的时间常数为 $L/(R+r)$，其值非常小，而时间常数变小会使应答速度加快。

(2) 将串联的二极管和稳压二极管一起并联到电机中(图中的(b))，此与第 2 章所述的继电器和螺线管线圈的电路具有相同的原理。



(a) 续流二极管　　(b) 稳压二极管+续流二极管

图 9.5 电涌电流的吸收

9.2 步进电机的驱动

9.2.1 微处理器与接口

将微处理器和计算机用于步进电机的控制中具有很多优点，它使控制顺序是可编程的，各种控制行为按时序进行，容易实现软件反馈等等。

为了使微处理器或计算机能与步进电机有机地结合，必须配备 I/O 端口和电流驱动电路。在 I/O 端口中使用专用的外围 LSI。可以选择的产品主要有：

(1) Z-80 系列的 PIO,8086 系列的 PPI(如 8255A 等)。

(2) 68 系列 PIA(MC 6821 等)。

9.2.2 2 相励磁的程序

2 相励磁的连接方法如图 9.6 所示[11]。A、$\overline{A}$、B 和 $\overline{B}$分别与 8255A 的端口 B 的各端 PB0～PB3 相对应。端口 B 的输出从右向左看,用斜线部分表示 1,空白部分表示 0 时,步 1 的输出是 1001,即 0x09。因此,与步进时间对应的输出如下:

电机的步进时间:1　　2　　3　　4　　…

端口 B 的输出:　0x09　　0x05　　0x06　　0x0a　　…

当电机反转时,步进时间的顺序为...,4,3,2,1。

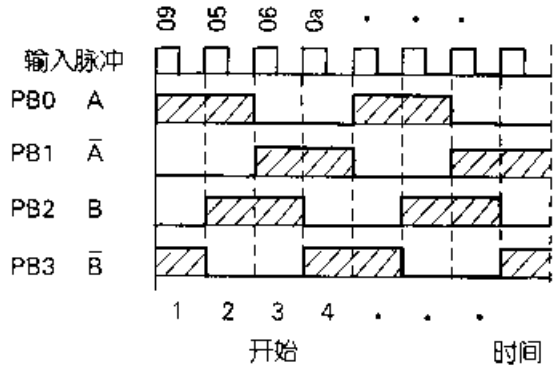


图 9.6　2 相励磁方式

以下是电机正、反转的程序示例。

```
/* HB 155.C use 8255 A-No.2, port B */
# include <stdio.h>
# include <dos.h>
# include <conio.h>
# define PA2 0x0328              /* 8255A  的地址 */
# define PB2        0x032a       /* 最低位列由厂家指定 */
# define PC2        0x032c
# define CR2        0x032e
# define CW2          0x80       /* 8255A 端口的初始化:全输出 */
# define CCA          0x09       /* 图 9.6  PB0～PB3 的输出脉冲模板(pattern) */
# define CCB          0x05
# define CCA_     0x06
# define CCB_     0x0a
```

```
void movcw (int cc);                    /* 正转 cc 周 */
void wait (int times);                  /* 函数 wait */
void main (void)
{
  int c;
  outp(CR2, CW2) ;                      /* 8255A 初始化 */
  outp(PB2, 0);                         /* 电机停止 */
  while(! kbhit( ))                     /* 按键开始执行 */
  {
    movcw(300);
    wait(10000);                        /* wait( )已用秒表测试 */
  }
  outp(PB2, 0);                         /* 动作停止 */
}
void movcw(int cc)
{
  int mp;
  for(mp = 1; mp<= cc; mp + + )
  {
    outp(PB2, CCA);     wait(1200);
    outp(PB2, CCB);     wait(1200);
    outp(PB2, CCA_);    wait(1200);
    outp(PB2, CCB_);    wait(1200);
  }
}
    //反转时应逆转函数 movcw( )
```

9.2.3 专用驱动 IC

此处介绍的"TD62803P"是步进电机专用的简易驱动 IC,该芯片与 TTL 兼容,输出电流为 400mA,最高输出电压为 28V 或更大,适合于 3 相、4 相小型步进电机使用。图 9.7 是使用 TD62803P 的步进电机驱动电路示例[8,43,44]。电路的输出电压和电流在 12V、400mA 以下,可直接驱动电机,如果需要超过该限度时可借助电流容量更大的功率晶体管进行电机驱动。左边的输出部分是 8255 的 PB 输出端口。

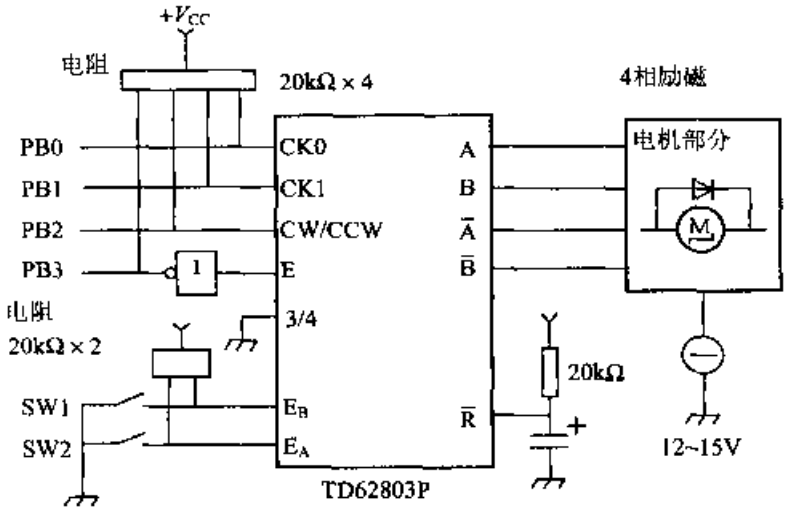


图 9.7 步进电机驱动电路[a]

(1) 利用 EA 和 EB 端选择 3 相、4 相电机的 1 相励磁、2 相励磁或 1-2 相励磁方式。例如,选择 4 相电机的 2 相励磁方式时对应如下数据组:

(EA,EB,3/4 端子)=(H,L,L)

(2) 按 PB3=E 端、PB2=CW/CCW、PB1=CK1、PB0=CK0 的方式连接。

(3) E 端本身用 H 导通励磁电流(ON),用 L 使其截止(OFF),同图中的 PB3 为 L 时使 E 端为 H,PB3 为 H 时使 E 端为 L。

启动电机:用 PB3 为 0x0 使 E 端为 H

停止电机:用 PB3 为 0x08 使 E 端为 L

(4) 正转、反转时 PB0～PB2 有表 9.1 所示的关系。在 4 相电机中,若 CW/CCW(PB2)端为 L 则向 CW 方向旋转,若为 H 则向 CCW 方向旋转。

此外,也有其它的驱动 IC 产品,如 TA7774P/F(双极型 2 电路驱动)等。

表 9.1 驱动的逻辑

PB2 CW/CCW	PB1 CK1	PB0 CK0	动 作
L	H	H 时转动 L 时保持	011(0x03)时 CW 010(0x02)时保持
H	H	H 时转动 L 时保持	111(0x07)时 CCW 110(0x06)时保持

(5) 以下是将 TD62803P 连接到 8255A 的端口 B,驱动电机正转、反转的示例代码,附带有关脉冲输出的振幅、间隔和函数 wait 的相关说明。

```
/* 正、反转程序 */
# include <stdio.h>
# include <dos.h>
# include <conio.h>
# define PA          0x0280                    /* 端口地址 */
# define PB          0x0282
# define PC          0x0284
# define CR          0x0286
# define CW          0x90                      /* PA:输入,PB:输出,PC:输出 */
# define CWP0        0x02                      /* 保持 表9.1 */
# define CWP1        0x03                      /* 正转 同上 */
# define CCWP0       0x06                      /* 保持 */
# define CCWP1       0x07                      /* 反转 */
# define MOFF        0x00                      /* 电机停止 */
# define TIMEP       20                        /* 适当的脉冲振幅 */
# define TIMER       250                       /* 适当的脉冲间隔 */
void movcw(int port, int tb);
void movccw (int port, int tc) ;
void motoff(int port);
void wait(int times);                          /* 函数声明 */

void main(void)
{
    int countb ;                               /* 计数变量 */
    outp(CR,CW) ;                              /* 8255的初始化 */
    outp(PB,CWP0) ;                            /* 电机启动准备 */
    countb = TIMEP;                            /* 脉冲间隔 */
    while(! kbhit( ))                          /* 有按键时返回 */
      movcw(PB,countb) ;                       /* 运转 */
    motoff(PB) ;                               /* 停止 */
}
    void movcw(int port, int tb)               /* 正转函数 */
{
    outp(port, CWP1) ; wait (TIMEP) ;
    outp(port, CWP0) ; wait(tb) ;
}
void movccw(int port, int tc)                  /* 反转函数 */
{
    outp(port, CCWP1) ; wait (TIMEP) ;
    outp(port, CCWP0) ; wait (tc) ;
}
void motoff(int port)                          /* 停止函数 */
```

```
{
    outp(port, MOFF) ;
}
void wait (int times)          /* 等待函数 */
{
    int i;                     /* 注:因为 time 已在库中定义故使用其它名字 */
    for(i=0; i<times; i++)
      printf("") ;
}
```

9.3 步进电机的加速和减速控制

9.3.1 分频比

1. 分频比

分频比(PulseRate)R_p的含义是在多少个基本振荡脉冲(频率f_0)中能够产生1个驱动脉冲电机的脉冲。利用R_p可将脉冲电机的旋转频率q表示为:

$$q = \frac{f_0}{R_0} \text{ [pps]} \tag{9.1}$$

称q为步进速率。

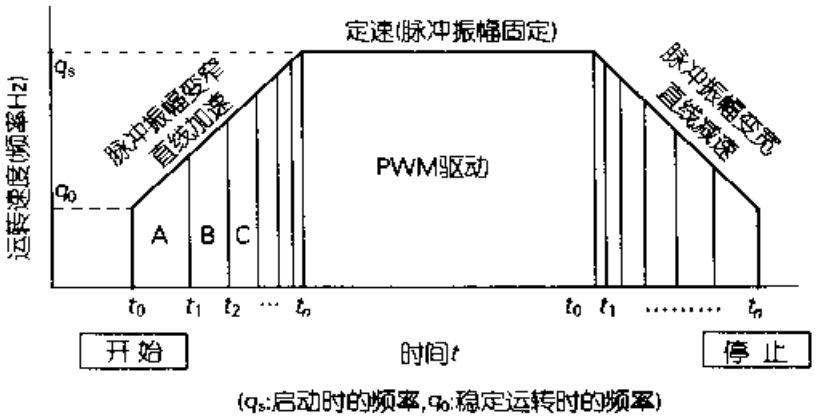


图9.8 步进电机的加速和减速

Ampel公司的PPMC-101允许将实际用的分频比设置在2～255之间,若

内部时钟不超过f_0 = 100kHz,则电机转动频率为 49～625Hz。当然,如果加上100kHz 的外部时钟,则实际可用的转动频率约变成原来的 8 倍。由于电机输出频率大约为 5kHz,因此,100kHz 的时钟频率基本属于上限。

2. 加速的必要性

加速脉冲数根据电机的种类、负载的移动惯量等的不同而有很大差异。负载的移动惯量较大时,加减速脉冲数需要相应变大,使电机缓慢地加速或减速,此即所谓的**慢启**和**慢停**方法。因此,电机运转时按如下顺序进行(参见图9.8):

低速启动→加速(慢启)→定速运转→减速(慢停)→停止

通过软件和硬件都可以实现电机的加速和减速。

此外,电机一般有共振点,即使转矩为 0 时频率也不为 0,所以,从启动到跳过共振点这一阶段必须保持高速运转。

9.3.2 加减速的控制方式

1. 直线加速

图 9.8 中给出了速度指令(步进速率)与脉冲计时之间的关系,其假定条件如下:

(1) 假定用脉冲序列 0～9 为止的 10 个脉冲进行加速。

(2) 面积 A、B、C、…是每个时间步内脉冲的积分

(3) 假定加速度为 α、脉冲计时为t_i、脉冲间隔是 Δt_i

(4) 步进速率,即每 1s 的步进数用q、启动时用q_0(Hz,pps)表示。

(5) 通过速率(最高脉冲数/s)用q_s(pps)表示。

此时,有:

$$q = g + \alpha t, \quad g = q_0 - \frac{\alpha}{2q_0} \tag{9.2}$$

用从t_0到t_n的脉冲围成的面积在第n步相等的方法计算出t_n。转矩的计算方法可参考书后的参考文献[27,45]。

9.3.3 加减速范围[43,45]

从 0 号脉冲开始到$(n-1)$号脉冲位置做直线加速,加速度 α 经计算得到。第n号脉冲以后为常速运转范围。下式给出了加速度的计算方法:

$$\text{加速度 } \alpha = \frac{2(q_s^2 - q_0^2)}{\sqrt{(2n-3)^2 + (q_s/q_0)^2 - 1} + (2n-3)} \tag{9.3}$$

可见,若给定q_s、q_0和n即可计算出加速度 α,并可求出脉冲计时t_i。

以下讨论对按常速步进速率q_s运转的电机进行直线减速。通常,减速度 β

应大于加速度 α、脉冲间隔较小为宜。若共用了 m 个脉冲使电机转动停止，停止时的步进速率假定为 q_1，则减速度可根据下式计算：

$$\text{减速度 } \beta = -\frac{2(q_s^2 - q_1^2)}{\sqrt{(2m-1)^2 + (q_s/q_1)^2 - 1} + (2m-1)} \tag{9.4}$$

详细设计可参考书后参考文献[45]。

9.3.4 加减速过程

首先需要将上节的计算结果做成**数据表**以供查询。在加速、定速、减速子例程内，对每一个脉冲输出，参照数据表进行查询，得到时间间隔数据。取出间隔数据的方法可以用软件定时方法或使用间隔定时器。

```
void main( )
{
  static int table0[50] = {150, 90, 60, 45, 35, 27, ...};
                          /* 准备若干个表 */
  kasoku(table0, n);      /* 参照表 table0 内的 n 个数据 */
  teisoku(table1, k);     /* 以下同 */
  gensoku(table2, m);
}
```

9.3.5 实际的步进电机控制

1. 小型步进电机

典型的小型步进电机实例是 Coparu 公司的 SP 4-415[46]，规格为 12V、4相、12 极、步进角 15°，每周转动为 12 步，自启动频率 250 pps，最大连续应答频率为 320 pps。

2. 驱动电路

专用驱动电路 DM-402X 的时钟 CP 最高可达 1500 pps，可以使用 8255 生成脉冲信号。

3. 控制器 LSI

Interface 公司的 4 轴步进电机控制器"IBX-7204"[47]，配备有 Ampel 公司的 PPMC-103A(使用 12MHz 的单片机)，输入采用光电绝缘，输出采用 40mA 的集电极开路，脉冲输出速度在通常的模式下为 98～1270 pps，最高可达到 16 000 pps。从计算机端只需要给出命令和参数即可实现常速驱动、梯形驱动和减速停止等，控制简单。

控制信号主要包括 CW/CCW 切换、极限开关和基准点输入等等。Oriental Motor 公司的驱动器"UDX 系列"是一款设计较为适合的产品，CosmoSystem 公司和日本脉冲电机公司也销售同样的控制器。此外，2 相励磁用的驱动 IC 可以选用 PMM8723(山洋电气)及 TA9415P(东芝)等产品。

9.4 DC 电机的驱动

9.4.1 控制用电机

作为传动装置的主要部分，电动机在我们的身边比比皆是。例如，用在传真机、打印机和复印机中的前述的步进电机，用于洗衣机中的可逆电机，用于电气计时器中的同步电机，用于吸尘器和搅拌机中的交直流两用万能电机以及用于自动调焦照相机中的超音速电机等等。

以前，电机只是简单地转动并产生能量，现在，对应于具体状况，不只旋转角和旋转方向，甚至旋转速度和加速度也可以控制，已发展到了**智能电机**的时代。机电一体化中最重要的一环就是通过计算机对电机进行随需控制。

1. DC 电机

这是利用直流供电的电机。在电机中，利用永磁材料或电磁材料构成外侧的圆筒部分，称为定子(stator)，其内部有 3 极铁心，每极外部都缠以线圈，构成转子(armature)。为了让电流能够通过转子，设置有 3 段换向器。DC 电机的内部结构如图 9.9 所示。由于 DC 电机具有体积小、转矩大和具有线性特性等优点，常被用于机器人控制等领域。不过，因为 DC 电机需要保留电刷和换向器，在最近的工业机器人中，已逐渐被不需要这些附加设备的 AC 电机所取代。

2. 无刷电机

将 DC 电机中的换向器和电刷取出后的结构如图 9.10 所示，转子设计为圆筒形的旋转磁铁。用光电孔可检测出转子的角度位置，通过电流交变在定子中生成旋转磁场。这种电机可以进行高速旋转且不需要换向器，因此，VTR、盒式走带机构、CD 播放器及 FDD 的驱动部分都使用它。

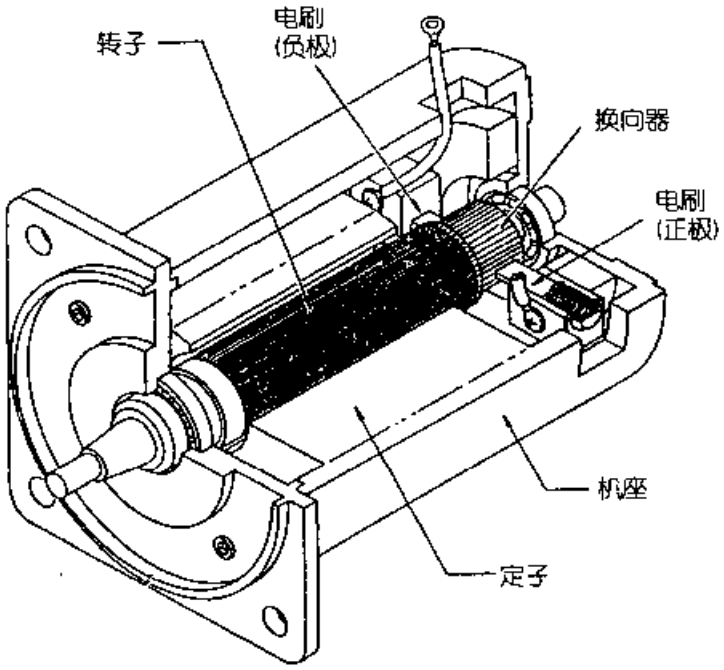


图 9.9 DC电机的内部结构[48]

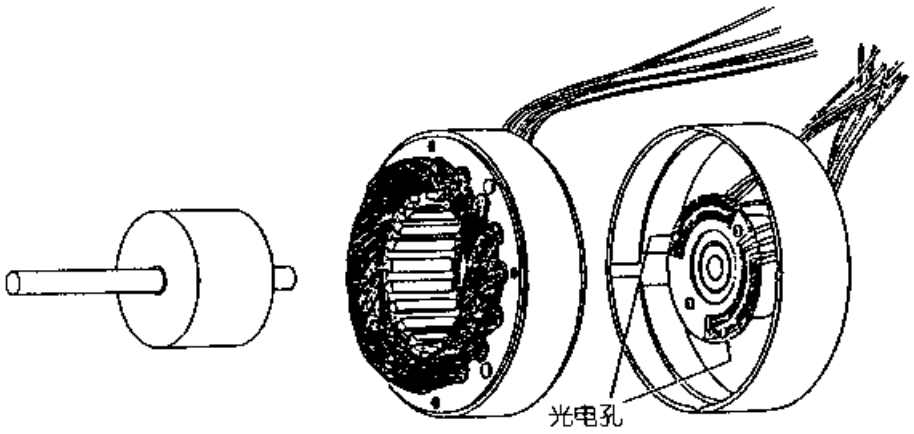


图 9.10 无刷电机的内部结构[48]

9.4.2 小型 DC 电机的控制 IC

在 FDD、HDD、8mm VTR 和 OA 机器等方面广泛使用只有数瓦功率级别的小型 DC 电机，因此，适合有刷电机、无刷电机和步进电机等使用的单片机型专用驱动 IC 也非常多，初学者可以很方便地买到并使用满足自己要求的电机功率的专用驱动 IC。

所谓**桥式驱动** IC，是因为驱动输出晶体管以非常类似 H 的形状排列而得名。使用这种 IC 可以控制有刷小型电机的正转、反转、制动和停止 4 种状态(模式)。

桥式驱动 IC 可分为二种类型：

(1) H 开关型——用逻辑(开关量)输入控制上述 4 种模式的切换。通过改变驱动电机的电源电压调节转速，利用输出电压控制端子改变输出电压。

(2) H 放大型——采用线性控制。用输入电压的大小和正负控制电机的转速和方向的改变。

电机制动是利用感应电势实现的短接制动(short brake，也称为再生制动)，即当切断供给电源的同时，将电机两端之间的电阻短路。在 H 开关型商品 IC 中，有 TA7267 BP(1 个回路、工作电压 18V、电流 1A)以及 TA7257 P、TA7291 P、TA 7279 P(2 个回路)、TA 7272 P、TA 8406 P 和 TA 8407 等。

9.4.3 H 开关型桥式驱动电路

图 9.11 为手持式 TA 8429 H，该产品价格低，体积小，最大电源电压为 30V，平均输出电流为 3.0A，可以驱动机器人用的数十瓦的电机。同图中的(b)给出了连接示意图，表 9.2 说明了其工作方式。电机速度通过 PB1 及 PB2 的脉冲振幅变化来控制(PWM 控制)。

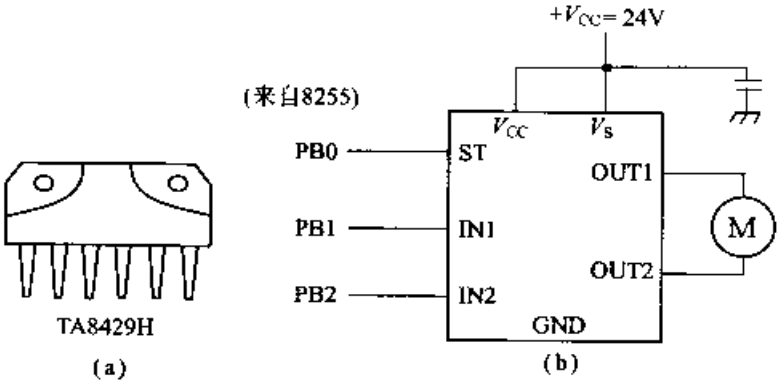


图 9.11 H 开关型 DC 电机驱动器

表 9.2 TA8429 的功能

输 入			输 出		工作模式
IN1	IN2	ST(stand by)	OUT1	OUT2	——
H	H	H	L	L	制动
L	H	H	L	H	反转
H	L	H	H	L	正转
L	L	H	OFF		停止
any	any	L	OFF		stand by

9.4.4 H放大型(线性)桥式驱动电路

图9.12为手持式TA 7272 P(2个回路),此驱动电路的额定电压为±18V(通常限制在±15V左右),输出电流为1.2A,可驱动10W左右的小型电机。同图的电路中使用电位计进行调节,形成逐渐加大目标值的电路。将电路的输出端与小型电机直接相连时,可实现如下操作:

(1) 用DA转换器的输出电压(目标值)可任意调节速度和旋转方向。

(2) 电压为0时电机停止运转。

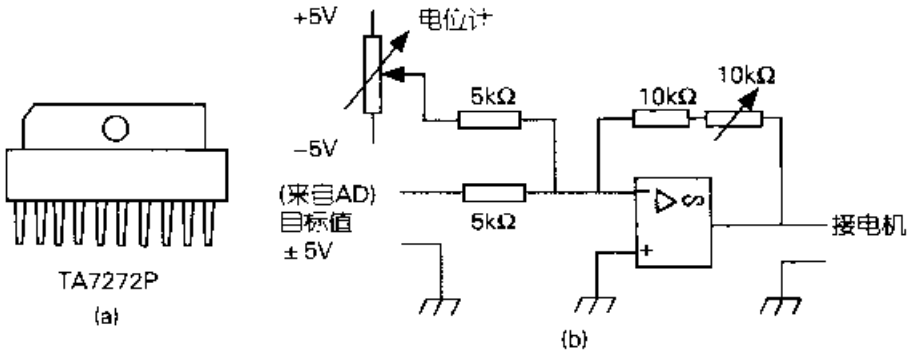


图9.12 H放大型DC电机驱动器

9.4.5 电机(电气系统)和负载(机械系统)的结合

DC电机具有线性转矩特性,可用于高精度的自动控制。通常,小型电机中的定子用永磁材料、转子端外缠以线圈,而线圈本身的阻抗很小可忽略不计。

1. 电气部分

根据图9.13,有:

电机的发生转矩 τ=(转矩常数k_t)×(转子电流i)

输入电压v =(电枢电阻R)×(电流i)+(反向感应电势e) (9.5)

反向感应电势e =(电势变比k_e)×(旋转角速度ω)

由此可得到转矩为:

$$\tau=\frac{k_t(v-e)}{R} \tag{9.6}$$

其中的转矩常数k_t可以从电机的产品目录中查到,电势变比k_e通常与k_t相同。

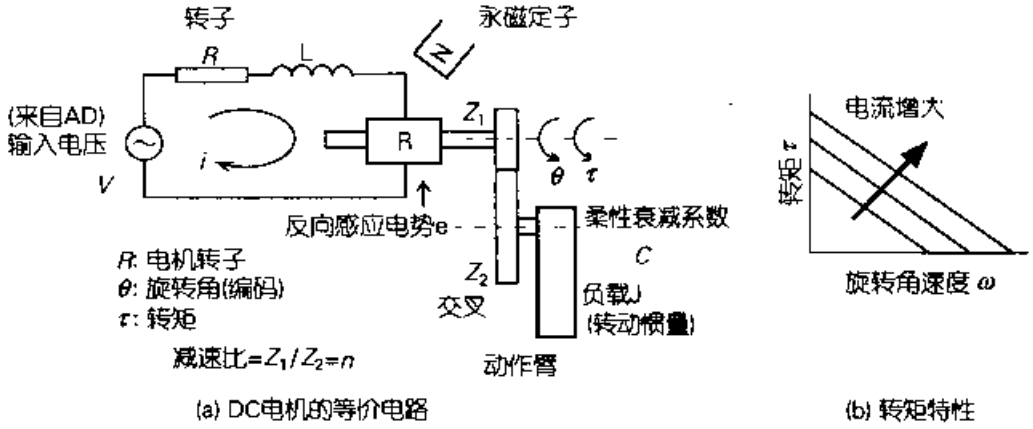


图 9.13 DC 伺服电机的工作和转矩特性

2. 机械部分

$$J\frac{d\omega}{dt}+C\omega=\tau \tag{9.7}$$

其中,J 为负载的转动惯量,C是黏性衰减系数,θ 是旋转角,$\omega=d\theta/dt$ 。负载包括机器人的动作臂、负载的物品、车轮或砝码等。

9.4.6 PLL 和 PWM

1. PLL 控制

这种控制方法先检测出作为基准的频率和应答频率之间的相位差,通过控制使频率、相位与检测出的基准值相吻合。这种方法主要用于电机的速度控制中,产品有 TC 9242 P 等专用 IC。

2. PWM 控制

这是一种通过控制电机的电流驱动的持续时间长短来提高效率的控制方法,常用于伺服电机驱动器以及变温空调和照明器具等方面。

9.5 用计算机控制 DC 电机

这里给出的是使用计算机控制 DC 伺服电机的实例,包括伺服机构的构成和 1 轴机器臂的位置控制方法,以及有关控制程序的说明。

9.5.1 数字伺服机构

近年来,随着计算机技术的发展,用在机械系统控制等方面的计算机(微机

等)越来越多。由于计算机内部处理的信息都是数字量,因此,将有计算机系统介入的伺服机构称为数字伺服机构。在控制领域利用计算机主要有如下优点:

(1) 一系列的动作可以用程序控制连续执行。

(2) 通过程序的变化,可任意设置控制系统的特性。

(3) 可以根据工作目的和工作环境的状态改变控制系统的特性。

图 9.14 中给出了数字伺服机构的构成示例。例中,从伺服电机的输出轴引出的电位计输出的模拟信号经 AD 转换成数字信号,连同从旋转编码器来的数字信号被输入到计算机。在计算机内部,将输入的实测值与程序给出的目标值进行比较、运算,结果经 DA 转换输出。该指令值(电压)被输出到伺服放大器(伺服驱动器),经放大后即可驱动伺服电机。

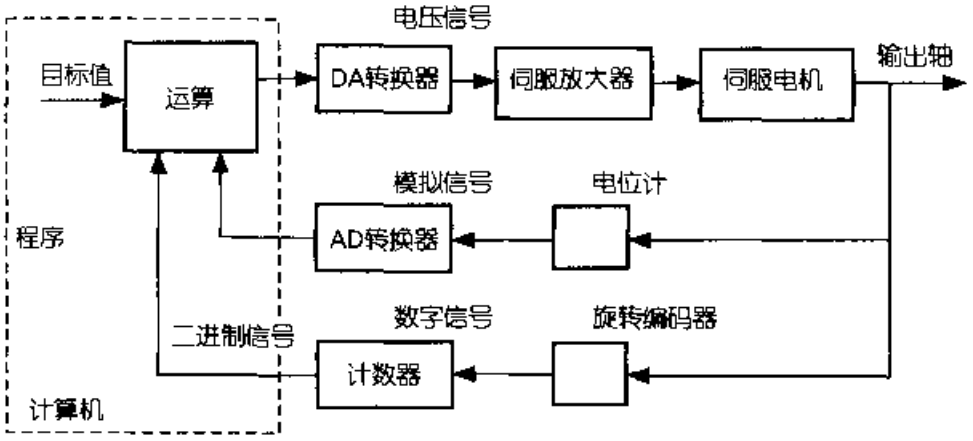


图 9.14 数字伺服机构的构成示例

9.5.2 简单的电机反馈控制

DC 电机中最基本的控制是对旋转速度的控制。这种情况下,在电机轴上最好按上旋转速度传感器(转速传感器)(近来更多的是将对编码器的输出进行数值微分作为旋转速度,从而取代了转速传感器)。从 AD 转换器读入转速传感器的值,与设定的速度目标值比较,以增加或减少施加给电机的输出(DA 转换器的电压)。当然,应该预先测定好电机速度和转速传感器的输出之间的关系。

```
DAvolt = 1;                 /* 施加给 DC 电机的 DA 转换器电压 */
Taco0 = s0;                 /* 作为目标的转速传感器值 a0 */
Tc = Read_taco( );          /* 用 AD 转换器读取转速传感器的值 */
DAvolt += alpha(Taco0 - Tc);
```

```
/* 用转速传感器的偏差(目标值 - 实测值)的 α 倍 */
/* 增减 DA 转换器的输出电压 */
```

其中,假定用比例因子 α 所确定的比例进行控制,用与角度的目标值和实测值的差(偏差)成比例的控制输入施加给 DC 电机。

在角度变化控制中,需要附加直接连接电机的旋转编码器(这种商品编码器很多,可直接买到),对此角度目标值和编码器的实测值的差实行反馈。这种用角度偏差和旋转速度偏差进行适当的加(减)法运算进行控制的方法称为 **PD 控制**(比例微分控制)。为了使控制更平稳,通常将 I 成分做加(减)法运算(根据试验误差得到适当的积分值),从而构成了同时控制位置和速度的更具有一般性的 **PID 控制**。

9.5.3 软件伺服控制基础和速度曲线

如前所述,数字伺服机构中,利用计算机控制对象的一系列动作过程可依赖程序连续实现,且用户可以很容易地通过改变程序来任意设置控制系统的特性。这里,我们就机械系统的工作过程的确定方法给以示例说明。

通常,在机器人等机械系统的控制场合,考虑到对其工作内容和周围以及系统自身的影响时,必须尽量减少振动和冲击。因此,需要控制其滑动时的加减速,从而控制其运行速度,此时一般使用具有以下凸形特征的速度曲线[49]。

(1) 单程的定义。往、返的动作各是一个单程,可以将 2 个单程曲线组合来考虑。

(2) 时间 t 增加时,用位移 s 的增加确定曲线的形状(给出 s 的单调增加的表达式)。

(3) 速度曲线中只含有一个峰值,加速度曲线在前半部为正,后半部为负。

使用具有凸形特征曲线时,将时间 t 、位移 s 、速度 v 、角速度 α 以及对角速度微分得到的跃变 j 无量纲化为(T,S,V,A,J)来考虑。而凸形曲线的特性值包括无量纲速度 V 、无量纲加速度 A 和无量纲跃变 J 以及它们的最大值 V_m、A_m和 J_m。

以下是关于几种凸形曲线的说明。

1. *摆线曲线*(Cycloidal Curve)

摆线曲线是加速度曲线用 1 个周期的正弦曲线表示的曲线。此曲线适合高速、轻负载情况,对振动不敏感,缺点是A_m较大,V_m和转矩也较大。

此外,摆线曲线的数学性质非常简单,通常作为一种理论分析的基本曲线使用。时间T和位移S的关系由下式给出:

$$S = T - \frac{1}{2\pi}\sin 2\pi T \tag{9.8}$$

2. 变形梯形曲线(Modified Trapezoid Curve)

摆线曲线的缺点是A_m较大,为了降低此曲线的峰值,可考虑使用正弦系列的变形梯形曲线,它将原曲线的形状近似为梯形。此曲线的特征是A_m较小,同样适合于高速、轻负载的情况。在控制对象的加速度最大值相同的情况下,与摆线曲线相比,变形梯形曲线可以有约 13%的加速。各常数由下式表示:

$$\left.\begin{aligned}
&T_a = \frac{1}{8}\\
&A_m = \frac{1}{1/4 - T_a + (2/\pi)T_a}\\
&V_a = \frac{2T_aA_m}{\pi}\\
&S_a = \frac{2T_a^2A_m}{\pi} - \frac{4T_a^2A_m}{\pi^2}\\
&V_b = A_m(0.5 - 2T_a) + V_a\\
&S_b = \frac{A_m(0.5-2T_a)^2}{2} + V_a(0.5-2T_a) + S_a\\
&S_c = 1 - S_b\\
&S_d = 1 - S_a
\end{aligned}\right\} \qquad (9.9)$$

且时间 T 和位移 S 的关系可用区间表示为:

区间 Ⅰ $(0 \leqslant T \leqslant T_a)$:

$$S_a = \frac{2T_aA_mT}{\pi} - \frac{4T_a^2A_m}{\pi^2}\sin\left(\frac{\pi T}{2T_a}\right)$$

区间 Ⅱ $(T_a \leqslant T \leqslant 0.5 - T_a)$:

$$S = \frac{A_m(T - T_a)^2}{2} + V_a(T - T_a) + S_a$$

区间 Ⅲ $(0.5 - T_a < T \leqslant 0.5 + T_a)$:

$$S = \frac{4T_a^2A_m}{\pi^2}\left\{1 - \cos\frac{\pi(T - 0.5 + T_a)}{2T_a}\right\} + V_b(T - 0.5 + T_a) + S_a$$

区间 Ⅳ $(0.5 - T_a < T \leqslant 1 - T_a)$:

$$S = \frac{-A_m(T - 0.5 - T_a)^2}{2} + V_b(T - 0.5 - T_a) + S_c$$

区间 Ⅴ $(1 - T_a < T \leqslant 1)$:

$$S = \frac{4T_a^2A_m}{\pi^2}\left[\cos\left\{\frac{\pi(T - 1 + T_a)}{2T_a}\right\} - 1\right] + V_a(T - 1 + T_a) + S_d$$

(9.10)

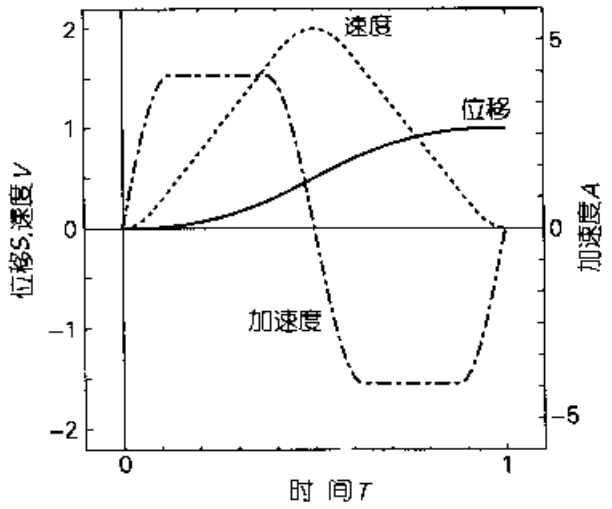


图 9.15 变形梯形曲线

图 9.15 中曲线给出的位移 S 、速度 V 以及加速度 A 可表示成无量纲时间 T 的函数。

3. 变形正弦曲线(Modified Sine Curve)

变形梯形曲线适合于高速轻负载的情况，因其V_m较大而不能够适应重负载。对于重负载的情况，希望有V_m较小的曲线。变形正弦曲线即是此类曲线之一，它在满足曲线的连续性和A_m较低的同时，V_m也较前述曲线为小。这种曲线有非常好的对称性，曲线光滑，且在负载的性质未知时使用最不容易出危险。各常数可表示如下：

$$\left.\begin{aligned} T_a &= \frac{1}{8} \\ A_m &= \frac{1}{2T_a/\pi + (2-8T_a)/\pi^2} \\ V_a &= \frac{2T_aA_m}{\pi} \\ S_a &= \frac{2T_a^2A_m}{\pi} - \frac{4T_a^2A_m}{\pi^2} \\ S_b &= 1 - S_a \end{aligned}\right\} \tag{9.11}$$

且时间T 和位移S 的关系表示为：

$$
\left.\begin{aligned}
&\text{区间 I} \quad (0 \leqslant T \leqslant T_a):\\
&S = \frac{2T_aA_m}{\pi}T - \frac{4T_a^2A_m}{\pi^2}\sin\frac{\pi T}{2T_a}\\
&\text{区间 II} \quad (T_a < T \leqslant 1-T_a):\\
&S = -\frac{(1-2T_a)^2A_m}{\pi^2}\left\{1-\cos\frac{\pi(T-T_a)}{1-2T_a}\right\}\\
&\quad + V_a(T-T_a)+S_a\\
&\text{区间 III} \quad (1-T_a < T \leqslant 1):\\
&S = \frac{4T_a^2A_m}{\pi^2}\left\{\cos\frac{\pi(T-1+T_a)}{2T_a}-1\right\}\\
&\quad + V_a(T-1+T_a)+S_b
\end{aligned}\right\} \tag{9.12}
$$

图 9.16 中给出了利用此曲线表示的位移S、速度V 以及加速度A 。

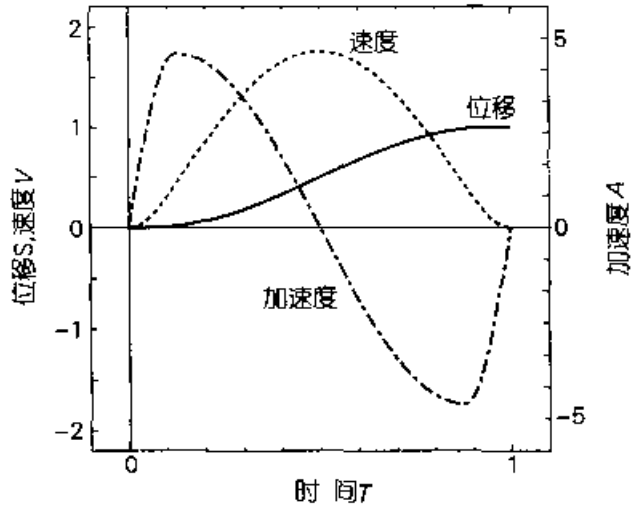


图 9.16 变形正弦曲线

以上的凸形曲线已经无量纲化，将其用于实际控制时，要设置达到最大位移S_m的时间T_m，用 C 语言编制对每个采样时刻的目标位移、速度以及加速度进行计算的程序，从而实现对计算机的控制。

9.5.4 使用速度曲线进行位置控制

这里使用前述的凸形曲线，如图 9.17 所示，进行 1 轴机器臂的位置控制。机器臂的各参数包括臂的重量M = 1.19kg，转动中心和臂的重心的距离L = 0.145m，相对于臂的重心的转动惯量J = 0.0507kg · m^2。在执行装置中，使用备有调波驱动减速机（减速比 1/50）的 DC 伺服电机（Harmonic Drive Sys-

tems 公司 RH 20-6003,输出为 35W),在伺服驱动器中使用 ServoLand 生产的伺服模量 SMCM-6 AI。此外,在电机轴上连接光学编码器(Incremental,使用 4 倍频电路,每周 2000 个脉冲的计数板)。计算机采用 NEC 公司的微机 PC-9821 Bp(CPU 80486 Dx2,66MHz)。接口板采用第 3 章所示的 32 位计数板及第 6 章所示的 DA 板。

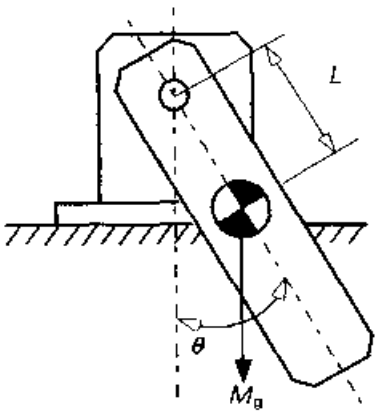


图 9.17 1 轴机器臂模型

控制器按图 9.18 框图所示,由适合计算转矩控制的前馈循环和前述的适合 PID 控制的反馈循环构成。这里,第 i 次采样时的角度指令值 θ_{ri}、加速度指令值 $\dot{\theta}_{ri}$ 及角加速度 $\ddot{\theta}_{ri}$ 可以利用前述的凸形曲线计算出来。实际的角度 θ_i 可以由驱动电机的旋转编码器测出,再利用下式计算在采样时刻 T_S 时的数值微分,即可得到角速度 $\dot{\theta}_i$ 以及角加速度 $\ddot{\theta}_i$:

$$\dot{\theta} = \frac{\theta_i - \theta_{i-1}}{T_S}, \ddot{\theta} = \frac{\dot{\theta}_i - \theta_{i-1}}{T_S} \tag{9.13}$$

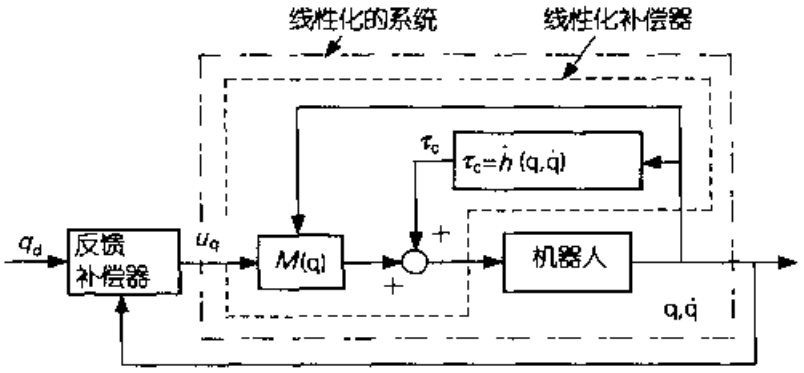


图 9.18 框图(计算转矩控制 + PID 控制)

利用已计算来的状态量(θ_i,$\dot{\theta}_i$,$\ddot{\theta}_i$),加上实验装置的各种参数,即可按如下方法计算出发送给电机的控制输入转矩u'(Nm):首先,当输入转矩为T 时,1 轴机器臂的运动方程为:

$$J\ddot{\theta} + ML\dot{\theta}^2 + MgL\sin\theta = T \tag{9.14}$$

其中,左边第 1 项为惯性转矩,第 2 项为离心力,第 3 项为重力转矩。黏性摩擦转矩忽略不计。如果给定臂的各参数J、M及L ,控制输入转矩$u(t)$为:

$$u(t) = J\ddot{\theta}_r + ML\dot{\theta}_r + MgL\sin\theta_r \tag{9.15}$$

电机为了驱动臂必须产生$u(t)$大小的转矩。因为公式中使用的指令值(θ_r,$\dot{\theta}_r$,$\ddot{\theta}_r$)是根据计算得来的,所以称为计算转矩,此相当于图 9.18 所示的前馈循环。不过,在实际控制中由于有模型化误差和外界因素的影响,还要附加前述的 PID 控制等反馈循环。综合考虑,控制输入转矩$U(t)$应由下式确定:

$$U(t) = u(t) + K_p(\theta_r - \theta) + K_d(\dot{\theta}_r - \dot{\theta}) + K_i\int(\theta_r - \theta)\mathrm{d}t$$

这里,K_p、K_d及K_i分别为比例、微分及积分反馈因子。

以下是使用凸形曲线(摆线曲线)实现 1 轴机器臂的位置控制程序的主要部分。因为只要使用 1 个备有减速机的伺服电机即可以构成单轴的控制臂,通过下述程序以及工业机器人中广泛使用的凸形曲线,可以亲身体会软件伺服控制过程。

```
/* 各参数的设定 */
# define CNT2RAD      (PI/2000/50)          //计数器的分辨率(rad/pulse)
# define M      1.19                        //臂的重量(kg)
# define L      0.145                       //臂的重心位置(m)
# define J      0.0507                      //臂的转动惯量(kgm^2)
# define XI     0.0735                      //电机的转矩常数(Nm)
# define G      9.80665                     //重力加速度
# define PI     3.14159                     //圆周率
...
extern void timer_intrinit(Ivoid *)func, double st, int cl);
                                            //定时器中断子函数
//func:中断程序;st:采样定时器;cl:系统时钟
...
void main( )
{
    ...
    timer_intrinit(pid_control, stime, CLOCK);    //定时器中断设定
    while(flag == CONTROL_ON)                     //中断终止条件
      timer_inrrest( );                           //定时器中断结束
    ...
```

```
}
void pid_control( )
{
    ...                          /* 旋转角度、加速度、角加速度的计算 */
    outport(STROBE, 0);                                    //向计数板输出锁存选通信号
    count1.io_data.low_data = inport(COUNTER1_LOW);
                                                           //读入计数器 1 通道的低位字节
    count1.io_data.high_data = inport(COUNTER1_HIGH);
                                                           //读入计数器 1 通道的高位字节
    th = -(double)count1.long.data * CNT2RAD;
                                                           //计算转角θ
dth = (th-pre_th)/stime;                                   //计算角速度θ̇(数值微分)
ddth = (dth-pre_dth)/stime;                                //计算角角速度θ̈(同上)
                                                           // stime:采样定时器
/* 凸形曲线(指令值)的计算——摆线曲线 */
thr = (t/T_MAX-1.0/2.0/PI * sin(2.0 * PI * t/T_MAX)) * S_MAX;
                                                           //角度指令值
dthr = (1.0-cos(2.0 * PI * t/T_MAX)) * S_MAX/T_MAX;
                                                           //角速度指令值
ddthr = (2.0 * PI * sin(2.0 * PI * t/T_MAX)) * S_MAX/T_MAX/T_MAX;
                                                           //角加速度指令值
/* PID 控制 */
eth = thr * S_MAX-th;                                      //计算角度偏差
edth = dthr * S_MAX/T_MAX-dth;                             //计算角速度偏差
sth + = eth;                                               //计算角度偏差的微分值
motor = (int)((J * ddthr + M * G * L * sin(th))/XI +       //计算转矩项
            Kp * eth + Kd * edth + Ki * sth;               //反馈项
/* 输出到 DA 板 */
outport(DA_CH1, motor);
...
}
```

图 9.19 和图 9.20 分别是非连续曲线的步进应答和利用摆线曲线做凸形曲线的位置控制的实验结果示意图。图示说明，虽然步进应答表面上平滑地逼近目标值，但控制开始时转矩指令值非常大，且在停止时会产生残留的振动。同时也会发现，即使采用最简单的摆线曲线，也会使控制转矩减小且抑制振动，可以实现平滑的位置控制。

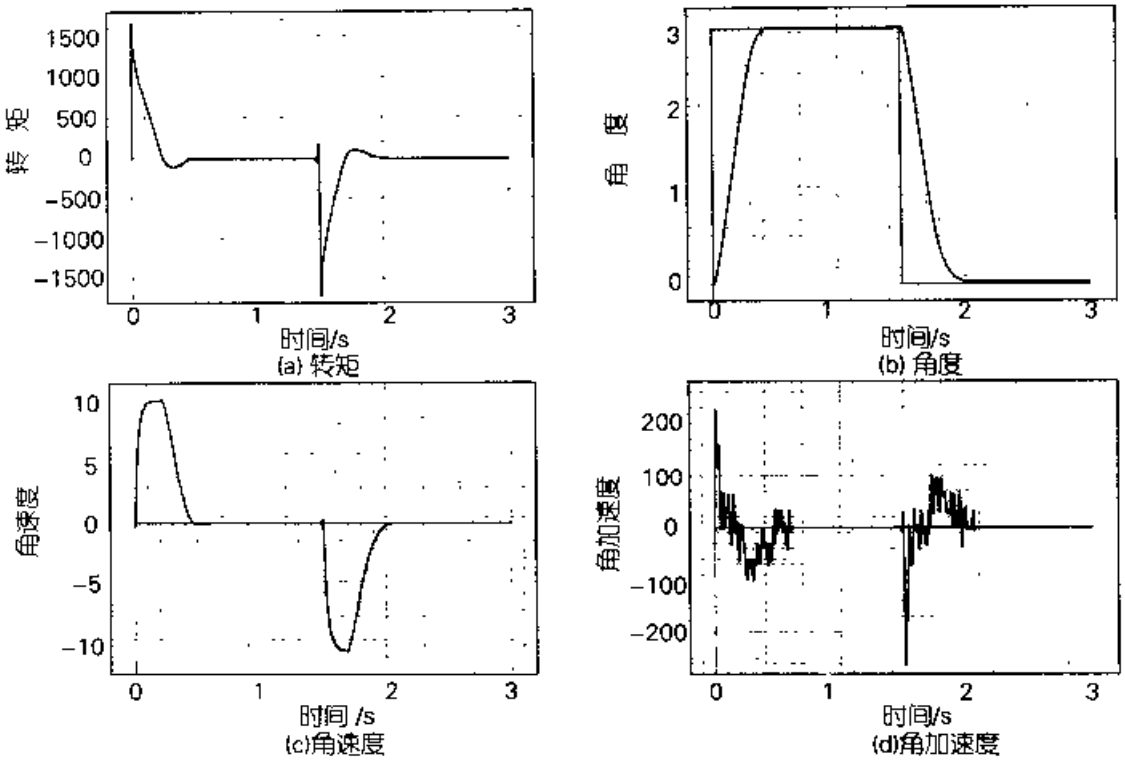
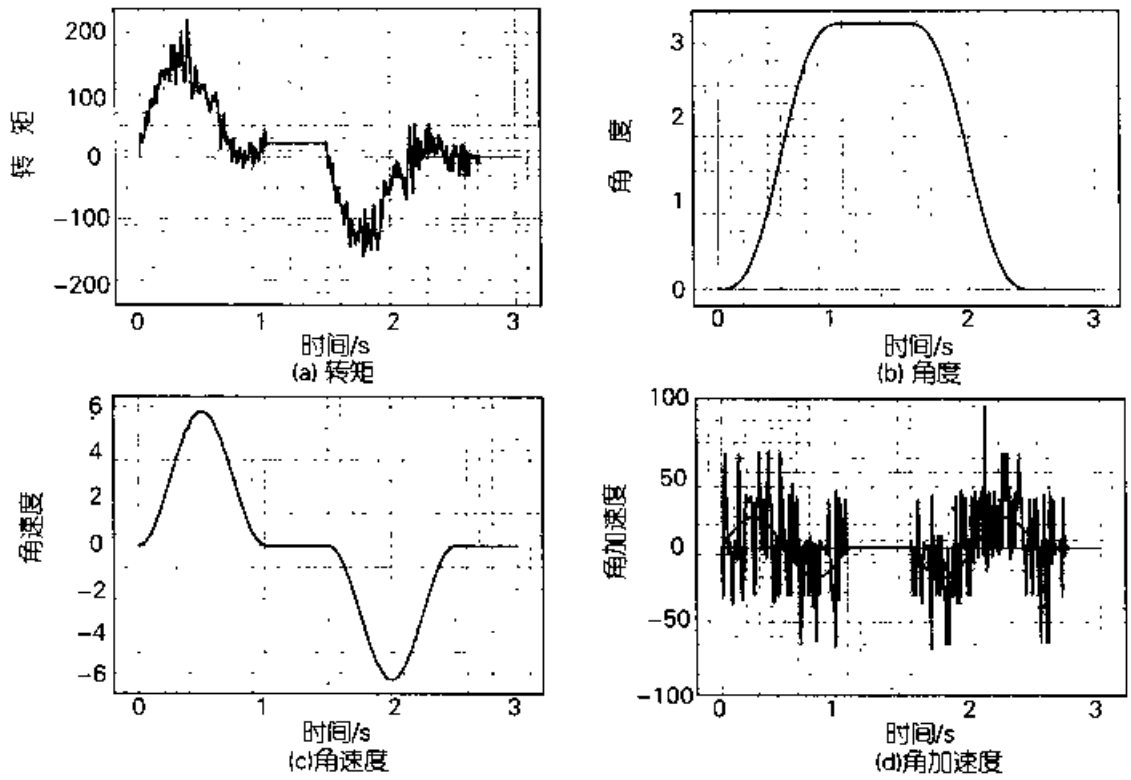


图 9.19 步进应答实验结果



(采样时间:0.001s,反馈因子:Kp=36000.0, Kd=600.0, Ki=1.0)

图 9.20 使用摆线的实验结果

练习 9

问题 9.1 假设有一个含有达林顿线圈的步进电机的驱动电路，可以用 8255A 的端口 B 驱动，其 2 相励磁的连接方式如 9.21 所示，有：

PB3:A, PB2:$\overline{A}$, PB1:B, PB0:$\overline{B}$

已知图中给出的为正转方向的时序，请依此编制实现电机正转的函数 mcw( )。

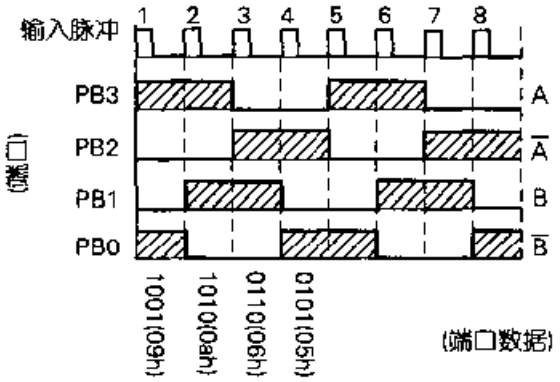


图 9.21 时 序

问题 9.2 假定有一个步进电机，每个脉冲的转角为 3.6°，使用 TD62803P 的专用驱动 IC，并采用 9.2 节中所述的连接方式：

PB3 = E, PB2 = CW/CCW, PB1 = CK1, PB0 = CK0

正、反转时，CK1 保持 H，脉冲被加到 CK0，从而切换 CW/CCW 端子。即：

正转：PB 的 bit 为 0000 00011

反转：PB 的 bit 为 0000 00111

电机停止：0x08

开始时，电机正转 1 周停止，此后，按一次键则反转一周。试编制满足此要求的程序。

问题 9.3 步进电机启动时的脉冲频率为 500pps，采用直线加速，试计算在第 10 个脉冲时达到 1500pps 并进入常速运转的脉冲计时[45]。

问题 9.4 说明 DC 电机的优缺点。

附录

附录 A　多操作系统

用 SYSTEM Commander 4(Soft Board 公司)实现多操作系统共存

在 DOS(FAT-16)中,每个基本分区最大只允许 2GB,但可以设置几个这样的基本 DOS 分区。例如,可以将磁盘划分为 4 个分区,分别为 2GB 的 DOS GB、2GB 的 Windows 98、2GB 的扩展分区和 2GB 的 Linux。事实上,由于最多只能划分为 4 个分区,如果有 3 个为基本分区,剩余的一个即是扩展分区。

有几种设置硬盘分区的方法,此处所述主要是使用 FDISK(FAT-16 BASE)进行磁盘分区的过程。

(I) 用命令 A:>format C: /s 从软盘将硬盘格式化。

(ii) 为了能用命令 A:\setup 将 MS-DOS 安装到 C:盘,先从 FDD 系统启动并将 C:盘划分出一个 2GB 的分区:

1) 执行 FDISK 命令,在 C:盘划分一个最大为 2GB 的 MS-DOS 分区。如果原磁盘上有内容,可能需要先删除原来的分区,然后再重新分区。

2) 执行命令 A:sys C:将 MS-DOS ver.6.2 的系统文件传送到硬盘,然后执行 FDISK 将该分区设置为活动(Active)的,使系统可以从该分区引导。磁盘上的其它空间暂时不用处理。

这是第一个 OS(FAT-16)。应该说明,生成新的基本分区之前必须要删去以前的扩展分区,故需要将原来的内容做必要的备份。

(iii) 用 A:\setup 命令安装 SYSTEM Commander(SC)。

1) 安装后,执行 C:\sc>SCIN 命令,并选择"可以使用 SC"参数。

2) 将 SC 复制到 E 盘等作为备份(C:\>xcopy C: E: /s),若不能拷贝也可以在以后重新安装。

(iv) US 模式:执行 C:\dos\us 或 chev us 命令。

1) 执行 C:\sc\scdisk 命令,并选择"change boot for OS instali"参数,输入 9,再输入 1。

经过上述处理后,系统已不能再从第一个基本 DOS 分区引导,但同时开放了从第二个隐含分区的引导能力。执行 FDISK 命令,系统显示:

```
1          NON DOS      1000                    第 1 个分区被抑制
C:2    A   PRI DOS      2047     FAT16          第 2 个分区变成活动分区
```

(v) 现在生成第 2 个基本分区。用 FDISK 命令确定第 2 个基本分区的大小(2GB 以内),并设置其为活动分区,然后插入 DOS 系统盘并从软盘启动。

1）执行 A:\>format C: /s 命令，然后用 A:\>sys C:传入系统文件。利用 FDISK 可确认第 2 个分区是活动的，且是 FAT-16 文件系统。

2）取出软盘并重新启动系统。

(vi) 将 Windows 98 安装到 C:盘。不能从启动盘安装时，可以直接从 CD-ROM 安装：

1）D:\win98\setup　　;D:为 CD-ROM

2）利用 FOISK 命令确认 C:Windows 分区为 FAT-16 文件系统

(vii) 用前述的备份恢复 SC 或启动 Windows，在“运行”菜单项窗口中指定文件名重新安装 SC：

1）重新安装 SC 到 Windows 系统所在的驱动器。

2）输入命令 C:\sc>SCIN，选择“可以使用 System commander”。

3）关闭 Windows 并重新启动。

(viii) 用 FDISK 命令生成扩展分区(不超过 2GB)，并选择扩展驱动器为 D:，将其格式化。

(ix) 重新启动系统，出现 SC 的启动菜单。

1）启动 MS-DOS 时其工作驱动器为 C:，此时是完全的 MS-DOS ver.6.2，且 Windows OS 的工作驱动器是 E:。因为都采用 FAT-16 文件系统，所以，从 DOS 中也可以正常操作 Windows 的文件。

2）相反地，若启动 Windows，则 Windows 的工作驱动器是 C:，而 DOS 则为 E:。

(x) 按此方式安装 DOS 后，config.sys 和 autoexec.bat 文件的设置方法都与原来的标准 DOS 相同。因此，在启动 DOS 后，MS-VC 和 TC++ for DOS 都可以正常工作，至于程序的编译，则与 3.4.2 节、3.4.3 节所述的 DOS 窗口中的操作完全相同。这样，用 C 语言编写的中断程序和 ESC 序列等都可以执行了。

现在，就步骤(ⅲ)之后如何再划分一个分区进行说明，但此方法仅适用于 Windows NT 而不适用于 Windows 98。

(iv′) 安装完 SC 并再次启动系统、出现 OS 选择菜单时：

1′) 按下 Alt+O 启动 OS Wizard。

2′) 选择 OS Wizard 的 Cancel 后，再选择上面按钮中的“partition(P)”，则会出现 HDD 的 Cylinder Group。

3′) 选取 Resize 按钮修改第 1 个 FAT 分区的大小(最大 2GB)并重新启动。

v′) 生成第 2 个 OS 分区。

1′) 当重启后出现 OS 选择菜单时，再次按 Alt+O 键启动 OS Wizard。

2′) 在安装方法中选择“New Install”，并在 OS 的种类中选择“Windows 95/98/NT”。

3′）选择"Partition"后点击"Regular"，则生成 FAT-16 文件类型的分区。

4′）最后选择"Isolated by itself"，就生成了 Windows 自己的分区。

（vi′）现在安装 Windows NT、执行 FDISK 确定活动分区并安装 SC。若安装失败，返回到(vi′)或(v′)重新执行。

（vii′）扩展分区的生成方法也是先执行(iv′)的步骤 1)和 2)，然后按 Creation 按钮进行分配。

不过，如果使用近来的 Partition Magic(Net Japan 公司)或 Commander Premium Pack DV / HC(Softboat 公司)，可以很容易地将 FAT-32 和 FAT-16 的 4 个以上分区一起进行设置，且 FAT-32 文件系统可以分配 4GB 或 8GB 乃至更大的空间。

附录 B　DOS/V 用的显示控制例程

此处列出的程序清单包括清屏、光标的打开与关闭、光标的位置确定及等待输入等函数。程序中不必设置图形模式即可显示彩色文本。

```
// discon.h display control for DOS / V
# include <stdio.h>
# include <dos.h>
# include <conio.h>
union REGS inregs,outregs;
void clrscr (void)                    // 显示器清屏
{
    printf("\x1b" "[2J");
}
void clrtxt (void)                    // clear text screen
{
    printf("\x1b" " *");
}
void video_mode(int mod)              // 视频模式
{                    // mode = 3 : 80 × 25 color text
    printf("\x1b" " [%dh",mod);       // mode = 73 : 80 × 25 color text
}
```

```
                                    // mode 72 :  640×480,
                                       16-color graphic & 80×25 text

void locate(int x,int y)            // cursor position
{
    printf("\x1b" "[%d, %dH",x,y);
}
void cur_on(void)                   // cursor on
{
    printf("\x1b" "[>5l");
}
void cur_off(void)                  // cursor off
{
    printf("\x1b" "[>5h");
}
void ccolor(int color)              // 字符颜色: set text color
{
    switch(color)
    {
      case 0: printf("\x1b" "[30m"); break; // black
      case 1 : printf("\x1b" "[34m"); break; // blue
      case 2 : printf("\x1b" "[32m"); break;// green
      case 3 : printf("\x1b" "[36m"); break; // cyan
      case 4 : printf("\x1b" "[31m"); break; // red
      case 5 : printf("\x1b" "[35m"); break; // magenta
      case 6 : printf("\x1b" "[33m"); break; // yellow
      case 7 : printf("\x1b" "[37m"); break; // white
      case 8 : printf("\x1b" "[40m"); break; // gray
      case 9: printf("\x1b" "[44m"); break; // dark blue
      case 10: printf("\x1b" "[42m"); break; // dark green
      case 11: printf("\x1b" "[46m"); break; // dark cyran
      case 12: printf("\x1b" "[41m"); break; // dark red
      case 13: printf("\x1b" "[45m"); break; // dark magen
      case 14: printf("\x1b" "[43m"); break; // dark yellow
      case 15: printf("\x1b" "[47m"); break; // dark white
```

```
        default: printf("\x1b" "[37m"); break; // white
    }
}
int inkey(void)                    /* check and get character */
{
    return  bdos(0x06, 0xff, 0) & 0xff;
}
/* wait until key in;:暂停,直到有键盘输入为止 */
void push_key (int color)
{
    ccolor(color);
    cur_on( ); locate(25,1);
    printf("* * * 请按任意键. * * *");
    getch ( ); printf("\n");
    clrtxt( ); cur_on( ); ccolor(7);
}
unsigned char keyb_clear(void)                    // 清键盘缓冲区
{
    inregs.h.ah = 3;
    int86 (0x16,&inregs,&outregs);
    return outregs.h.al;
}
unsigned char keyb_status(void)                   // 读取是否有按键的数据
{
    inregs.h.ah =0x0b;
    intdos (&inregs,&outregs);
    return outregs.h.al;
}
unsigned char getShiftkey(void)                   // shift key
{
    inregs.h.ah = 2;
    int86(0x16, &inregs, &outregs);
    return (outregs.h.al);
}
```

附录 C　系统定时器的利用

在没有定时器板时使用此处的程序是很方便的。

```
/* vtimer 4.c DOS/V timer interrupt
  8254: contrl word addr PITC = 43h, counter #0 addr PIT0 = 40h
  int vector TVEC = 08h, timer mask TMS = 01h
  CW = 36 : counter #0, load LSB/MSB, mode 3, binary
  timer lk f0 = 1.19318 MHz; 0.8380965 [micro s]
  divisor = 65536 (0xffffh),
  f0/divisor = 18.2064819 Hz: 54.9254932 ms (about 55 ms)
  compiler MSVC(Microsoft Visual C++),>cl xxx.c
*/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#define PIC0v      0x20                /* DOS/V 8259 master PIC */
#define PITCv      0x43    /* DOS/V timer 8254 CW (control word) */
#define PIT0v      0x40                /* DOS/V tinier #0 address */
#define TVECv      0x08                /* DOS/V timer int vector */
#define TMASKv     0x01                /* timer mask */
#define RMASKv     0xfe                /* timer remove mask */
#define EOIv       0x20                /* non specific end of interrupt
*/
#define CWv        0x36                /* timer control word CW */
#define STime      54                  /* DOS/V minimum time [ms] */
int count;
void (_interrupt _far *origvec)( );    /* original vector */
void _interrupt _far Ftimer (void)     /* hook routine */
{
    count++;
    outp(PIC0v, EOIv);                 /* EOI */
```

```
}
void reset_timer(void)                                  /* reset timer */
{
    _disable( );
    _dos_setvect(TVECv, origvec);
    outp(PITCv, CWv);
    outp(PIT0v, 0);                                     /* divisor = 65536 */
    outp(PIT0v, 0);
    enable( );
}
void main( )
{
    int i, j, ncnt;
    printf("\nncnt ?");
    scanf("%d", &ncnt);
    _disable( );                                        /* hard int disable */
    origvec = _dos_getvect(TNECv);
    _dos_setvect(TVECv, Ftimer);
    outp(PITCv, CWv);
    outp(PIT0v, 0xc8);
    outp(PIT0v, 0x04);                  /* 1222 * 0.838 = 1.024 ms */
    _enable( );
    outp(PIC0v + 1, (inp(PIC0v + 1) & RMASKv));   /* enable timer */
    for (i = 0; i<ncnt; i++)
    {
      count = 0;
      while(count < STime);
      origveo( );
      outp(PIC0v + 1, (inp(PIC0v + 1) | TMASKv));/* disable timer */
      printf("%d\n", i);
      outp(PIC0v + 1, (inp(PIC0v + 1) & RMASKv));/* enable timer */
    }
      outp(PIC0v + 1, (inp(PIC0v + 1) | TMASKv));/* disable timer */
      reset_timer( );
}
```

附录 D　DA 转换器

此处所给出的程序包括 Interface 公司的中断例程和 Contec 公司的 DA 控制板操作例程以及如何调用这些例程的函数。使用 DA 采样时，若在 DA 之后马上进行 AD 转换，可以实现对每个数据的单独控制。

1. DA 的中断头文件

```
//irqdef 5.h ; dos/v interrupt for DA board ; DA 12-4(PC) (CONTEC Co.)
//and for IBX 3303 (Interface Co.)
//IRQ : 选自 3, 4, 5, 6, 7, 9, 10, if, 12, 14, 15
# include <stdio.h>
# include <dos.h>
# include <conio.h>
# define PIC0  0x20                    //master PIC
# define PIC1 0xa0                     //slave PIC
typedef unsigned int DATA;             //定义数据的类型
typedef unsigned char UCHAR;
unsigned int orgmsk[2];                //处理原始中断屏蔽状态
unsigned int irqtab[] = {              //中断向量号表
            0x00, 0x00, 0x00, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
            0x00, 0x71, 0x72, 0x73, 0x74, 0x00, 0x76, 0x77};
unsigned int msktab[]={                //中断屏蔽位表
            0x00, 0x00, 0x00, 0x08, 0x10, 0x20, 0x40, 0x80,
            0x00, 0x02, 0x04, 0x08, 0x10, 0x00, 0x40, 0x80};
//unsigned int intn=IRQN;              //选定的 IRQ 号
int IRQflag = 0;                       //中断用标志
void (_interrupt _far * orgvec)();     //原始中断向量定义
void IRQ_install(int intn, void (_interrupt _far * irqfunc)(void));
                                       //函数 int 的定义
void set_mask(int);
void remove_mask(int);
void IRQ_reset(int);
```

```
void IRQ_eoi(int);
//
//中断设置
// Inputs: intn——中断号(0～15)
// irqfunc——中断处理用的 far 指针
void IRQ_install(int intn, void (_interrupt _far *irqfunc)(void))
{
    _disable( );                                    //关中断
    orgvec = _dos_getvect(irqtab[intn]);//取原始中断向量
    orgmsk[0] = inp(0x21);                          //取原始中断向量屏蔽(主片)
    orgmsk[1] = inp(0xa1);                          //取原始中断向量屏蔽(从片)
    _dos_setvect(irqtab[intn], irqfunc);  //设置中断向量
    set_mask(intn);
    _enable( );                                     //开中断
    ccolor(3);
    printf("\norgvec %x", orgvec);      //函数 ccolor( )仅设置字符颜色
    ccolor(2);
    printf("\ nIRQn vect mask ～mask: %2d %2x %2x %2x",
        intn, irqtab[intn], msktab[intn], ～msktab[intn]);
    ccolor(7);
    gets(buf);                                      //等待输入
}

//中断屏蔽的设置
void set_mask(int intn)
{
    _disable( );
    if (intn < 9)
      outp(0x21, (inp(0x21) | msktab[intn]));              // 主片用
    else
    outp(0xa1, (inp(0xa1) | msktab[intn]));                // 从片用
    _enable( );
}

//撤消计算机的中断向量
```

```
void remove_mask(int intn)
{
    _disable( );
    if(intn < 8)
      outp(0x21, (inip(0x21) & ~msktab[intn])); // 取消主片的屏蔽
    else
    {
      outp(0x21, (inp(0x21) & 0xfb));          //slave connecting point
      outp(0xa1, (inp(0xa1) & ~msktab[intn])); // 取消从片的屏蔽
    }

    // init ISR(in service registor) clear; enable to accept int
    if(intn < 8)
      outp(0x20, (0x60 | (UCHAR)intn));
    else
    {
      outp(0x20, 0x62);                        //slave connecting point
      outp(0xa0, (0x60 | (UCHAR)(intn-8)));
    }
    _enable( );
}

// 恢复原中断
void IRQ_reset(int intn)                    // intn:中断号(0～15)
{
    _disable( );                            // 关中断
    set_mask(intn);
    _dos_setvect(irqtab[intn], orgvec);// 恢复到原始的中断向量
    outp(0x21, orgmsk[0]);          // 恢复原始的中断向量的屏蔽(主片)
    outp(0xal, orgmsk[1]);          // 恢复原始的中断向量的屏蔽(从片)
    _enable();                              // 开中断
}

// 执行 EOI
void IRQ_eoi(int intn)
```

```
{
    if(intn < 8)
      outp(0x20, 0x20);          //输出到计算机的 EOI(IRQ3~8 时)
    else
    {
      outp(0xa0, 0x20);          //输出到计算机的 EOI(IRQ9~15 时)
      outp(0xa0, 0x0b);
      if(! inp(0xa0)) outp(0x20, 0x20);
    }
    outp(ADR+0, 0x20);                    //输出到控制板的 EOI
}

void _interrupt _far IRQ_routine(void)    //用户的 IRQ 例程
{
    IRQflag = 1;                          //仅设置标志即返回
    IRQ_eoi(IRQN);
}
```

2. 设置 DA 和定时器设置头文件

```
//condal2.h : CONTEC DA 12-4(PC), DOS level
//Compile : Microsoft C ver.6.0 / Microsoft C/C++ ver.7.0, Large
Model
#include <stdio.h>
#include <dos.h>
#include <conio.h>

/*
typedef unsigned int DATA;      //定义 DA 数据的类型(12 bit DA)
DATA adr0;                  //nnchan for data << 4; data &= nnchan
DATA adr2 = 0;
DATA resall = 0x80;             //reset all
DATA restrig = 1;               //reset trigger status
DATA adr3 = 0;
DATA digout = 0;                //0x80: digital out mode; 0 : none
DATA outmod = 0;                //0: single chan mode, 仅一个通道输出
DATA trigmod = 0;               //0: int trig by timer, 0x20: ext trig
```

```
DATA timgate = 0;                    //0: timer stop, 0x10: timer start
DATA intn = IRQN;                    //0: no IRQ, 5: IRQ #5 (IRQ <= 7)
*/
//形成 IO recovery 的时间
// inputs : lp——IO recovery 的循环次数
void iowait(int lp)
{
    int i, dat;
    for(i=0; i<lp; i++)
      inp(ADR+1);
}
void DA_init(DATA outmod, DATA trigmod, DATA timgate, DATA intn)
                                          //DA 12-4 的初始化
{
    outp(ADR+0x02, 0x80);
    adr3 = adr3 | outmod | trigmod | timgate | intn;
    outp(ADR+0x03, (char)adr3);
}
void DA_outS(int ch, DATA dat)           //输出一个 DA 转换数据

{
      outpw(ADR+0, dat*0x10+ch);
}
void DA_outM(int nch, DATA dat[])        //sequential data[ ]  output
{
    int i;
    for (i=0; i<nch; i++)
      DA_outS(i, dat[i]);
}
int DA_status(void)
{
    int d;
    d = inp(ADR+1) & 0x80;               //1:DAC'ing, 0:end
    return d;
}
```

```
int trig_in_status(void)
{
    int dat;
    dat = inp(ADR+1) & 0x10;        // 1: for timer output occurrence
    return dat;                     // cleared by trig_reset_status( )
}
void trig_reset_status(void)
{
    outp(ADR+2, 1);
}
int extern _trig_sense(void)
{
    int dat;
    dat = inp(ADR+3) & 0x40;        // 0: trigger high
    return dat;
}
void reset_all(void)
{
    outp(ADR+2, 0x80);
}
unsigned int di_inp(void)           // 读取通用输入数据
{
    unsigned int dat;
    dat = inp(ADR+3) & 0x80;        // 取通用数据
    return dat>>7;                  // 0:DI is high
}
void do_out( )                      // 输出通用输入数据
{
    outp(ADR+3, 0x80);
}
// ..... sub timer .....
// timer clock(8254) = = 4MHz; clk0——>clk1——>clk2——>out
// mode = = 2; 0.25 * c0 * c1 * c2[us]
void tim_stop(void)                 // 停止计时器
```

```
{
    outp(ADR+3, (adr3 & 0xef));
}
void tm_start(void)                          //启动计时器
{
    outp(ADR+3, (adr3 | 0x10));              //计时器的 GATE 端置为 H
}
// void tm_init(long hz);                    //定时器的初始化
// Inputs : (long)hz…定时器输出频率
//(单位:Hz 1～50000L)
void tm_init(long hz)                        // 参照 IBX-3118(Interface 公
                                                司)等
{
    unsigned int t0, t1;
    long tm, wk, t, i, m, freq;
    tm = 4000000L/4L/hz;                     //用时钟 4MHz 的 4 分频计算 tm

    m = tm;
    //计算距 tm = n*t 最近的并且最小 t1 值,其中,t 的值为 8～65535,
    //n 的值为 2～65535
    for(i=4L;i<65536L;i++)
    {
      if((wk=tm % i)<m)
    {
      if((tm/i)<65536)
      {
          m = wk;
          t=i;
      }
      }
      if(! m) break;
    }
    t0 = (unsigned int)t & 0xffff;           //用求得的 t 作为 t0 计算满足 tm
                                               =t0*t1 的 t0、t1
    t1 = (unsigned int)(tm/(long)t0);
```

```
    if(t0 < t1)                    // 假设 t0 <= t1
    {
      t0 = t1;
      t1 = (unsigned int)(tm/(long)t0);
    }
    tm_stop();                     //定时器的 GATE 端置为 L
                                   //设置定时器
    outp(ADR+14, 0x34); iowait(10);
    outp(ADR+8, 4); iowait(10);
    outp(ADR+8, 0); iowait(10);
    outp(ADR+14, 0x74); iowait(10);
                                   // 计数器 #1
    outp(ADR+10, t0 % 256); iowait(10);
                                   // 计数器 #0
    outp(ADR+10, t0 / 256); iowait(10);
    outp(ADR+14, 0xb4); iowait(10);
                                   // 计数器 #1
    outp(ADR+12, t1 % 256); iowait(10);
    outp(ADR+12, t1 / 256); iowait(10);
    trig_reset_status( );
    tm_stop( );                    //定时器的 GATE 端置为 L
    freq = 4000000/4/t0/t1;
    printf("\ntime divisor : %d %d %d freq = %d",
            4, t0, t1, (int)freq);
    gets(buf);
}
    //此外,应注意使用 IBX-3118(Interface 公司)时与此头文件会有差别,
    //可参考附带的软件
```

3. 主函数

这里只给出了主要部分的示例。程序中利用 include 包括了前述的 2 个头文件。

```
// dasp 7.c : CONTEC dos/v DA board : DA 12-4 (PC)
// MS VC++ dos timer-interrupt for DA array data output
void main ( )
```

```
{
    int i, j;
    init_volt( );                          //输出电压置 0
    remove_mask(intn);                     //取消中断屏蔽
    trig_reset_status( );                  //复位触发器
    tm_start( );                           //启动定时器
    for(j = 0; j<mmdata; j + + )           //所指定的数据个数
    {
      while(! IRQflag);                    //等待中断
      IRQflag = 0;                         //若发生中断则清除标志
      trig_reset_status( );                //也清除触发器
      for(i = 0; i<nnchan; i + + )         //在各通道输出一个数据
        DA_outS(i, ddata[i][i]);           //此后可读入一个 AD 数据
    }
    init_volt( );                          //电压置为 0,停止定时器
    tm_stop( );                            //停止计数
    IRQ_eoi(intn);                         //IRQ 结束处理
    IRQ_reset(intn);
}
```

附录 E 图形应用程序

下述程序用于执行用户窗口的设置和图形字体的设置及显示等。

1. 视频控制头文件

```
// vidcon.h : video control for DOS/V
# include <stdio.h>
# include <dos.h>
union REGS inregs,outregs;
int get_videomode(void)                    // current video mode
{
    int d;
    inregs.h.ah = 0xf;
    int86(0x10, &inregs, &outregs);
```

```
    d = outregs.h.al;
    printf("\nVideomode = %d", d);
    return (d);
}
void set_gmode(int mode)          //640x480, 16 color, text 80x25

{
    switch(mode)
    {
      case 1: _setvideomode(_VRES16EXCOLOR);
              break;
      case 0: _setvideomode(_DEFAULTMODE);
              break;
      default: break;
    }
    _clearscreen(_GCLEARSCREEN);              //graphic clear
}
void set_gline(int lstyle)                    //line style
{
    _clearscreen(_GCLEARSCREEN);              //graphic clear
    _setviewport(0,0,639,479);                //phisical plane
    _setwindow(1, -1000., 1000., 1000., 1000); //chart scale
    switch (lstyle)
    {
      case 0:_setlinestyle(0xffff);
              break;                          //solid line
      case 1:_setlinestyle(0xc0c0);
             break;                           //broken line
      default: break;
    }
}
void end_gmode(int vmod)                      //reset G-mode
{
    char buf[20];
    gets(buf);
```

```
    if(vmod < 0)
      _setvideomode(_DEFAULTMODE);
    else
      _set_gmode(vmod);
    ccolor(7);
}
```

2. MS-VC for DOS/V(以及 MS-C)的图形

```
// f7.c MS-VC (MS-C) Graphics
// display : VGA color 640 × 480, 80 × 25 text
// compile : cl f7.c graphics.lib
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <graph.h>          // <graph.h> is for MS VC only, not for
VC + +
char buf[40];
# include "discon.h"                          // 前述的附录 D
# include "vidcon.h"
# define FONTA "c:\\msvc\\lib\\font"    // 存储图形字体的文件
//-------------------------------------------------------------
char * text[2] = {"Kanagawa-it"};
unsigned char * face[8] =
              {"t' helv'", "t' modern'", "t' roman'", "t' script'", "t' sys-
              tem'"};     // 字体(Font)
//-------------------------------------------------------------
int zone[2][4];                               // View port zone
void window0(int color, int zz[]);            // 分为 2 个区域(zone)
void window1(int color, int zz[]);
void window2(int, int);
void main (void)
{
    clrtxt( );
    clrscr( );
    _setvideomode(_VRES16EXCOLOR);  // set DOS/V 16 color
    zone[0][0] = 30;
```

```
    zone[0][1] = 0;                    // zone data
    zone[0][2] = 300;
    zone[0][3] = 100;
    zone[1][0] = 30;
    zone[1][1] = 130;
    zone[1][2] = 639;
    zone[1][3] = 479;
    window0(11, zone[0]);
    locate(3, 10);
    ccolor(4);
    printf("MS-VC Graphics");
    ccolor(7);
    window1(3, zone[1]);
    window2(5, 2);
    end_gmode(-1);
}
void window0(int color, int zz[])                    /* window0 setting */
{
    float wx0, wy0, wx1, wy1;
    _setviewport(zz[0], zz[1], zz[2], zz[3]); /* view port 0 */
    wx0 = -5. * (float) zz[2];
    wx1 =  5. * (float) zz[2];
    wy0 = -5. * (float) zz[3];
    wy1 =  5. * (float) zz[3];
    _setwindow(1, wx0, wy0, wx1, wy1);
    _setcolor(color);
    _rectangle_w(_GFILLINTERIOR, wx0, wy0, wx1, wy1);
    _moveto_w(wx0, wy0);
    _lineto_w(wx0, wy1);
    _lineto_w(wx1, wy1);
    _lineto_w(wx0, wy1);
    _lineto_w(wx0, wy0);
}
void window1(int color, int zz [] )                  // window1 setting
{
```

```
    //与 window0 基本相同
}

void window2(int color, int nfont)                    //显示图形字符
{
    int err;
    static unsigned char list[20];
    if((err = _registerfonts(FONTA"\\*.FON")) < 0)     //字体序号
    {
      printf("err1 %d", err);
      clrscr( );
      exit(0);
    }
    strcpy (list, face[font]);
    strcat(list,"h60w25b");                           //确定高度和宽度
    if(_setfont(list) >= 0)
      _setcolor(color);
    _moveto_w(0, 0);
    _outgtext(text[0]);                               //输出字符
    _setcolor(7);
}
```

3. TC++的图形

```
//f3.cpp 单元文件和画直线、设置背景颜色等。
//compile: tcc f3.cpp graphics.lib
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <graphics.h>                                 //TC++的图形库头文件
class graph0
{
    public:
    graph0( )
    {
      int gd = DETECT, gm;
      initgraph(&gd, &gm, "c:\\tc4\\bgi");            //初始化
```

```
 }
    ~graph0( )
    {
      closegraph( );          //结束处理
    }
    void line0(int x0, int y0, int x1, int y1)
    {
      line(x0, y0, x1, y1);
    }
};
int main( )                   //以下函数为 TC++ 标准库函数
{
    int i;
    graph0 g00;
    setbkcolor(11);
    setcolor(4);
    setlinestyle(1, 0, 3);
    g00.line0(0, 0, 639, 479);
    setcolor(1);
    setlinestyle(3, 0, 1);
    rectangle(100, 100, 400, 200);
    setcolor(6);
    ellipse(350, 220, 0, 360, 120, 60);
    getch( );
    return 0;
}
```


附录 F　鼠标器系统调用

1. DOS/V 的鼠标参数(INT 33h)(参照 196 页表)

2. DOS/V 机用的鼠标器操作头文件(PC-9800 系列基本相同)

```
/* mouse2.h
    1) 参考(河西：Microsoft C实践プログラミング入門, 技術評論社, p.230)
    2) devloe:MOUSE.SYS should be installed in config.sys
```

```
    3) mouse erase:modisp (0), display: modisp (1)
    4) mouse click process:
      free = 1;
      while(1)
      {
        moget(&right, &left, &x, &y);
        if(left ! = 0 && free = = 1)
        {
          ...;
          free = 0;
        }
        if(left = = 0)
          free = 1;
        }
    5) mouse_color: 0=blue, 1=red, 2=green
* /
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <stdlib.h>
static union REGS inregs, outregs;
static struct SREGS segregs;
static unsigned DSEG, GSEG;
void mouse_ini(void)      /* 初始化 * /
{
    inregs.x.ax = 0;
    int86(51, &inregs, &outregs);      /* 0x33 = 51 * /
    if(outregs.x.ax = = 0)
    {
      printf("mouse cant init !");
      exit(0);
      }
}
void mouse_disp(int flag)              /* mouse display/erase * /
```

DOS/V 的鼠标参数(INT 33h)

输入输出	寄存器			功　能
	ax	bx	cx/dx	
输入	0h			鼠标器初始化
输出				ax=0:没安装驱动程序
				ax=FFFFh:已安装驱动程序
输入	1h			显示光标
输入	2h			关闭光标
输入	3h			读取光标位置
输出		bit0		1=左键 ON,0=OFF
		bit1		1=右键 ON,0=OFF
		bit2		1=中间键 ON,0=OFF
			cx	光标的 X 坐标
			dx	光标的 Y 坐标
输入	4h			设置光标位置
			cx	X 坐标
			dx	Y 坐标
输入	5h			取某一键按下信息
		0h		左键
		1h		右键
		2h		中间键
输出	bit0			1=左键按下,0=未按下
	bit1			1=右键按下,0=未按下
	bit2			1=中间键按下,0=未按下
		bx		自上次调用以来被按下的次数
			cx	最后按下时的 X 坐标
			dx	最后按下时的 Y 坐标
输入	6h			取某一键放开信息
		0h		左键
		1h		右键
		2h		中间键
输出	bit0			1=左键放开,0=放开
	bit1			1=右键放开,0=未放开
	bit2			1=中间键放开,0=未放开
		bx		自上次调用以来被放开的次数
			cx	最后放开时的 X 坐标
			dx	最后放开时的 Y 坐标
输入	7h			设置鼠标器光标在 X 方向的移动范围
			cx	在 X 方向的移动范围的最小值
			dx	在 X 方向的移动范围的最大值
输入	8h			设置鼠标器光标在 Y 方向的移动范围
			cx	在 Y 方向的移动范围的最小值
			dx	在 Y 方向的移动范围的最大值
输入	bh			读取鼠标器的移动量
			cx	X 方向移动量
			dx	Y 方向移动量

```
{
    switch(flag)
    {
      case 0 :inregs.x.ax = 2;
              break;                          /* off */
      case 1 :inregs.x.ax = 1;
              break;                          /* on */
    }
    int86(51, &inregs, &outregs);             /* 51=0x33 */
}
void mouse_getpos(int *right, int *left, int *x, int *y)
                                              /* get mouse position */
{
    inregs.x.ax = 3;
    int86(51, &inregs, &outregs);
    *right = outregs.x.bx & 0x02;             /* right button */
    *left = outregs.x.ax & 0x01;              /* left button */
    *x = outregs.x.cx;                        /* x axis */
    *y = outregs.x.dx;                        /* y axis */
}
void mouse_button(int button, int flag, int *stat, int *count, int *x,
int *y)
                                              /* get mouse button */
{
    if(button == 'l' || button == 'L')        /* left button */
    switch(flag)
    {
      case 1 :inregs.x.ax = 5;
              break;                          /* push */
      case 0 :inregs.x.ax = 6;
              break;                          /* free */
    }
    if(button == 'r' || button == 'R')        /* right button */
    switch(flag)
    {
```

```
        case 1:inregs.x.ax = 7;
               break;                          /* push */
        case 0:inregs.x.ax = 8;
               break;                          /* free */
        }
    int86(51, &inregs, &outregs);              /* 0x33 = 51 */
    *stat = outregs.x.ax;                      /* push infom */
    *count = outregs.x.bx;                     /* push count */
    *x = outregs.x.cx;                         /* x coordinate */
    *y = outregs.x.dx;                         /* y coordinate */
}
void mouse_setpos(int x,int y)                 /* set mouse position */
{
    inregs.x.ax = 4;
    inregs.x.cx = x;
    inregs.x.dx = y;
    int86(51, &inregs, &outregs);
}
void mouse_style(int x, int y, char *buf) /* mouse style */
{
    inregs.x.ax = 9;
    inregs.x.bx = x;
    inregs.x.cx = y;                           /* cur center */
    segread(&segregs);
    segregs.es = segregs.ds;
    inregs.x.dx = (int)buf;                    /* style */
    int86x(51, &inregs, &outregs, &segregs);
}
void mouse_distance(int *x, int *y)        /* get move distance */
{
    inregs.x.ax = 11;
    int86(51, &inregs, &outregs);
    *x = outregs.x.cx;
    *y = outregs.x.dx;
```

```
}
void mickey(int x, int y)                    /* mouse sensitivity */

{
    inregs.x.ax = 15;
    inregs.x.cx = x;
    inregs.x.dx = y;
    /* micky/dot */
    int86(51, &inregs, &outregs);
}
void mouse_view(int xmin, int ymin, int xmax, int ymax)
                                             /* set mouse view area */
{
    inregs.x.ax = 16;
    inregs.x.cx = xmin;
    inregs.x.dx = xmax;
    int86(51, &inregs, &outregs);
    inregs.x.ax = 17;
    inregs.x.cx = ymin;
    inregs.x.dx = ymax;
    int86(51, &inregs, &outregs);
}
void mouse_color(int color)                  /* mouse color */
{
    inregs.x.ax = 18;
    inregs.x.bx = color;
    int86(51, &inregs, &outregs);
}
void mouse_delay(unsigned int t)             /* time delay */
{
    for (; t>0; t--);
}
```

3. 检测程序

```
// mstst2.c : 2D and 3D picture by mouse
# include <stdio.h>
```

```
# include "mouse2.h"
# include "dlscon.h"
void wait (long int);
void main ( )
{
    int xs, ys, x, y, xdum, ydum;
    int right = 0, left = 0, flag = 0, free = 1;
    int n, k, mx, my, num = 0;
    char c0;
    clrscr( );
    mouse_ini( );
    mouse_color(2);
    mouse_setpos(320, 200);
    mouse_disp(1);
    ccolor(3);
    while(left = = 0)
    {
      mouse_getpos(&right, &left, &x, &y);
      locate(10, 10);
      printf("%d", x);
      locate(10, 11);
      printf("%d", y);
    }
    left = 0; wait(50000);
    mouse_getpos(&right, &left, &xdum, &ydum);
    if(left > = 0)
      printf("\nend");
    mouse_disp(0);
    ccolor(7);
}
void wait(long int tt)
{
    while(tt- -);
}
```

练习题解答

❖练习 1

问题 1.1 解答略(可参考课文)

问题 1.2 机电一体化(mechatronics)是机械、电子及计算机 3 个领域相结合,利用传感器取得信息并利用计算机进行机械和设备控制的一门学科。设备名和组成结构略。

问题 1.3 并行 I/O、AD 转换器、DA 转换器和编码器(计数器)等控制板。使用方法的说明可从本书中找到。

问题 1.4 (1) 1/4W 左右的金属膜电阻。
(2) 10μF 以上的铝电解电容器。
(3) 0.001～0.1μF 范围的陶瓷电容器。
(4) 聚乙烯、聚丙烯等的薄膜电容器。
(5) 陶瓷(水泥)电阻或绕线电阻。
(6) 使用 10k～30kΩ 的陶瓷电位器。
(7) 稳压二极管和 OP 放大器。

问题 1.5 不同计算机的插头的形状和大小不同,若将软盘翻转则不能插入软盘驱动器等等。

问题 1.6 解答略(提示:可考虑如看护机器人和服务机器人等能够给予人类帮助的机器人方面)。

❖练习 2

问题 2.1 (1) 字节(Byte) (2) 255 (3) +127, −128 (4) NAND (5) 不可兼或,$X=\overline{A}\cdot B+A\cdot\overline{B}$

问题 2.2 参照图 2.A。

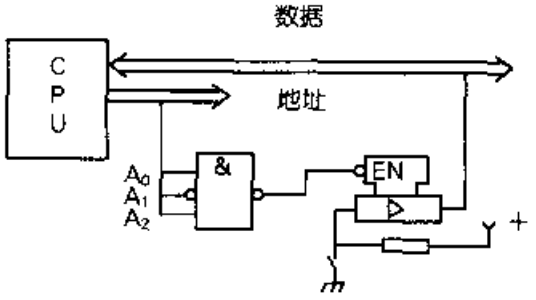


图 2.A 地址译码器示例

问题 2.3 (1) 0613　　(2) 由$(21)_{10}=(10101)_2=15h$,得 D615

问题 2.4 (1) 从 1000h 开始执行　　(2) 1000　　(3) 3E10　　(4) 用 10h 加载寄存器 A　　(5) 0601　　(6) 80　　(7) A + B→A　　(8) 从 A 中减去 PB　　(9) 寄存器 A 的结果是 10h + 01h − 03h = $(16+1-3)_{10}=1Eh$。

问题 2.5

```
                        PAD:  EQU   04h
                        PAC:  EQU   05h
                        PBD:  EQU   06h
                        PBC:  EQU   07h
;(地址)  (机器语言)
                              ORG   900h          ;程序的起始地址
0820     3E4F           INT:  LD    A,    4Fh     ;PAC 初始化
0822     D305                 OUT   (PAC), A     ;( )表示地址 PAC 中存放的内容
0824     3E0F                 LD    A,    0Fh     ;PBC 的初始化
0826     D307                 OUT   (PBC), A
0828     DB04                 IN    A,    (PAD)   ;输入
082A     D306           LOOP1: OUT   (PBD), A     ;原样输出
082C     17                   RLA                 ;寄存器 A 循环左移一位
082E     47                   LD    B,    A       ;保存 A
0830     DB04                 IN    A,    (PAD)
0832     0E             LOOP2: LD    C,    00h     ;与加载 255 相同
0834     0D             LOOP3: DEC   C             ;wait:C 变成 0 的时间
0836     C23408               JP    NZ,   LOOP3   ;循环 255 次
083A     3D                   DEC   A
083C                    C23208       JP    NZ,   LOOP2 ;计数结束
                        ;
0840     78                   LD    A,    B       ;恢复 A
0842     C32A08               JP    LOOP1
0844                         END
```

问题 2.6 解答略(参照图 2.3。将 CPU 的$\overline{IORQ}$连接到译码器 74LS139 的片选使能端$\overline{CE}$。此信号变为 L 时芯片选通。使用地址总线 A_7 和 A_6 作为 74LS139 的输入)。

问题 2.7 解答略。

❖练习 3

问题 3.1 (1)

```
#include <stdio.h>
#include <stdlib.h>
char fname[40];
double t[1024], x[1024];
void main( )
```

```
{
    FILE * fp0; int i;          /* 计算 t,x */
    scanf("%s", fname);
    if((fp0 = fopen(fname, "wb")) == NULL)  /* 二进制方式写 */
    {
      printf("\nerr!");  exit(1);
    }
    fwrite(t, 1024 * 8, 1, fp0);            /* double 类型数据占 8B */
    fwrite(x, 1024 * 8, 1, fp0);            /* 逐个写入记录 */
    fclose(fp0);
    if((fp0 = fopen(fname, "rb")) == NULL)  /*  按同一顺序读 */
    {
      printf("\nerr!");  exit(1);
    }
    fread(t, 1024 * 8, 1, fp0);
    fread(x, 1024 * 8, 1, fp0);
    fclose(fp0);
}
```

(2)解答略。

问题 3.2

(1)解答略

(2)解答略(参考课文中的字符操作)

(3)

```
#include <stdio.h>
#include <stdlib.h>
char file1[40] = "b1.tab";;
char buf[60];
void main( )
{
    FILE * fp10; float f1;
    if((fp10 = fopen(file1, "r")) == NULL)    /*  读文件 */
    {
      printf("\nopen_err!");  exit(1);
    }
    fscanf(fp10, "%f%s", &f1, buf);         /* 从文件中读出一行 */
    fclose(fp10);                           /* 空格为数据分隔符 */
    printf("\n%f %s", f1, buf);
}
```

问题 3.3

(1)

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
char buf[40];
double scan_double(char * ss, double r)          /* 数字变换函数 */
{
    printf("\n%s %lf", ss, r);
    gets(buf);
    if(buf[0] == 'Y' || buf[0] == 'y' || buf[0] == '\0')
                                        /* 按表示 yes 的键时什么也不做 */
      return r;
    else
      if(buf[0] >= 0x30 && buf[0] <= 0x39) /* 数字 0~9 */
      {
        r = atof(buf);
        return r;
      }
      else
        return-999.0;                          /* 意味着输入错误 */
}
void main( )
{
    double a;
    char ach[20] = "param1";
    a = scan_double(ach, 1.0);
    printf("\n%s %lf", ach, a);
}
```

(2) 这样的数据在使用 Excel 绘图时有用。使用语句：

```
fprintf(fp20, "%f %f\n", t[i], x[i]);
```

问题 3.4 (1) 解答略(提示：与 0xffdb 做 & 运算)。

(2) 解答略(提示：向左移 4bit 进行正、负判别。若为正与 0x08 做 | 运算，若为负则与 0xf7 做 & 运算。另一种解法是与 0x08 做 & 运算，若为 0 则 bit 3 置 1，为 1 置 0)。

问题 3.5 R[p1][p2] = α · A[p1][p2] · B[p2][p2]。

```
/* Matrix product;
    "square /non-square" R[][] = A[] * B[] * scol
    calling sequence;
    mat_product(A, p1, p2, B, p2, p2, R, p1, p2, scol);
* /
void mat_product(double * A, int p1, int p2,
```

```
                double * B, int p3, int p4,
                double * R, int p6,
                double scol)[1]
{
    int i, j, k;
    double s, C[Max * Max];
    if(scol == 0.)
      scol = 1.0;
    if(p2 != p3)
    {
      ccolor(4); printf("\nerr mat-prod");
      ccolor(7); exit(0);
    }
    for(i=0; i<p1; i++)
    {
      for(j=0; j<p4; j++)
      {
        s = 0;
        for(k=0; k<p2; k++)
          s += A[i*p2+k] * B[k*p4+j];
        C[i*p6+j] = s * scol;
      }
    }
    for(i=0; i<p1*p6; i++)
      R[i] = C[i];
}
```

问题 3.6 (1) 解答略。 (2) 解答略。 (3) 解答略。

❖练习 4

问题 4.1 (1) 在 MS-C 中为 d0 = inp(0x290);，Turbo-C 中为 d0 = inportb(0x290);。

(2) MS-C 中为 outp(0x282, d1);，Turbo-C 中为 outportb(0x282, d1);。

---

1) 原文中此函数定义形式如下：

```
void mat_product(double * A, int p1, int p2,
                 double * B, int p3, int p4,
                 double * R, int p2, int p6,
                 double scol)
```

函数中的参数 p2 重复，语法错误。考虑到后一个 p2 未用，故删掉。——译者注

问题 4.2 (1) 因为端口 A 为输入,端口 B 为输出,故 CW 应为 90h。按下 SW#0 和 SW#3 时对应写入 PA 的数据是 0x90。LED 的 0～2 号对应二进制数 00000111,即 07h。主函数内容如下:

```
#define CW 0x90
void main( )
{
  char d;
  outp(CR, CW);
  while(! kbhit( ))
  {
    d = inp(PA);
    if(d == 0x09)
      outp(PB, 0x07);          /* 若 SW=0x09 输出数据=7h */
    else
      outp(PB, 0);             /* 若 SW=0 输出数据=0 */
  }
}
```

(2) 解答略。

问题 4.3 (4) 0x90　　(9) 0x7　　(10) outp(PC, 0xff),其它略。

问题 4.4 (1) 外部电路 8255 的基本频率 $f_0 = 2.4576MHz$,周期为 0.4069×10^{-6}。此外,250Hz 为 4ms。

1ms 时必须的脉冲数:$n = 1 \times 10^{-3} / (0.4069 \times 10^{-6}) = 2458$

4ms 时必须的脉冲数:$n = 4 \times 10^{-3} / (0.4069 \times 10^{-6}) = 2454.6 \times 4 = 9830$

于是,高位字节=0x26,低位字节=0x66,CW 为:

① 使用计数器#1 时　　:$D_7 D_6 = 01$

② 按低位、高位的顺序加载　　:$D_5 D_4 = 11$

③ 模式 3,分频矩形波　　:$D_3 D_2 D_1 = 011$

④ 二进制计数　　:$D_0 = 0$

上述的位组合为 01110110,即 CW=0x76。

采样信号:用 OUT 信号作为输出到计算机的中断信号即可进行采样。

(2) 解答略。

问题 4.5 解答略(提示:用图和流程描述。(1)从 PA 输入的按钮#0 的数据是 0x01,按钮#1 的数据是 0x02,按钮#3 的数据为 0x08。(2)若有某个电磁开关为 ON 时,使其它开关为 OFF。此外,数字电路中也有所谓的优先电路)。

问题 4.6 解答略(提示:(1)使用反射型光电耦合器如 Omlon 公司的 EE-SF5B 等。这里,根据图 4.A 所示,在 5mm 的间隔 d 内可以检测出从白纸的反射来的光。来自 LED 的反射光被光电晶体管接收。用施密特触发缓冲器对波形整形,并用限幅器调整在 5V 范围内,输入到 8255A 的输入端口。(2)向 8255A 输入须状触觉开关或光电开关的 ON 和 OFF 信号,用计算机计数)。

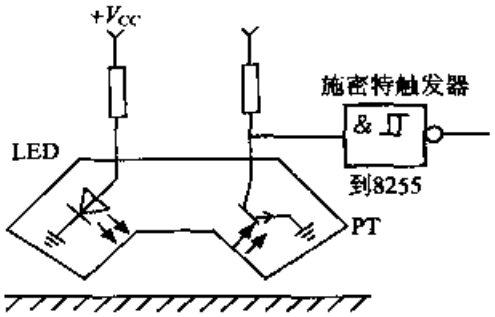


图 4.A　反射型光电耦合器

问题 4.7　使用译码器时程序较简单，当然也可以使用其它器件。此处采用译码器 IC (5068)。电路(图 4.B)和 C 语言程序如下：

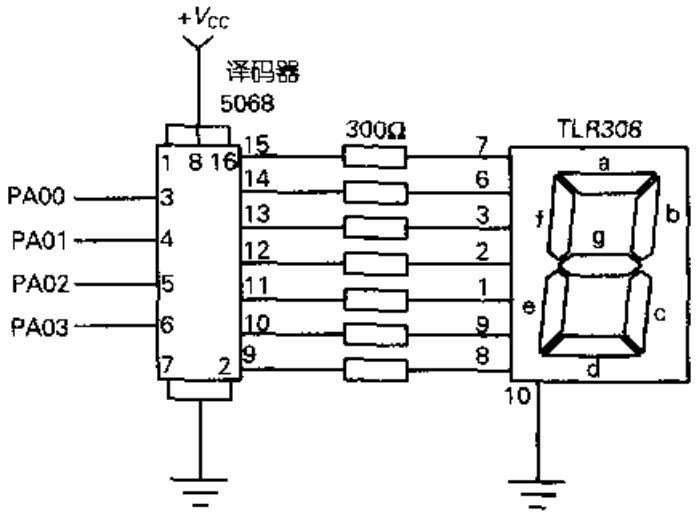


图 4.B　显示 16 进制数的 7 段 LED 电路

此程序的使用方法是：例如，当显示数字 3 时，在"Input number >[0～15]"的提示后输入 3 即可。

```
#include <stdio.h>
#include <dos.h>                    /* outport( )的头文件 */
#define BASE_ADDRESS 0x300
#define SETTING_ADDRESS (BASE_ADDRESS+3)
#define PORT_A (BASE_ADDRESS+0)
#define OUTPUT 0x80

void main( )
{
    int number;
    outport(SETTING_ADDRESS OUTPUT);      /* PIO 端口的初始化 */
```

```
    do
    {
      printf("Input number >[0～15]");          /* 指定要显示的数字 */
      scanf("%d", &number);
      outport(PORT_A, number);                  /* 输出到 PIO 端口 */
    }while(getchar() != 'q');
}
```

❖练习 5

问题 5.1 向量可参见课文中的表 5.1。若屏蔽数据从主 PIC、从 PIC 上开始分别为 0x1、0x2、0x4、…，则 IRQ3 是 0x08，IRQ5 是 0x20，IRQ10 是 0x04。

问题 5.2 解答略(参照课文)。

问题 5.3 使用 OCW2。DOS/V 的主 PIC 的地址由 PIC0(=0x20)定义时，非指定 EOI 为 outp(PIC0, 0x20);。

问题 5.4 执行_disable( )，接着 disable timer，打印后取消屏蔽。

问题 5.5 假定 CLK#0 和 CLK#1 的分频值(devisor:对时钟分频的分母)分别为 C0 和 C1(通常 C0<C1)。若设置 C0 为 8、C1 为 1000。则：

$f_0 \div C0 \div C1 = 1000Hz$

将此用 C 语言编码。

问题 5.6 通过"控制面板"中的"系统"程序进行观察时，若"声音控制器"的驱动程序不可使用则该卡不能正常工作。尽管取下声音控制卡并删除驱动程序该卡仍不能工作。找出原因，修改 BIOS 的设置(参照 7.1 节的(c))。

❖练习 6

问题 6.1 数字信号的特点：

① 信号的处理、运算和判断容易。

② 易实现高速化。

③ 数据的存储和传输容易。

④ 电路中不会再有散乱的元器件，易实现集成化和批量生产。

⑤ 噪声可能使系统崩溃。

⑥ 需要处理的元器件较多，消耗电量大。

问题 6.2 解答略(参考课文中的表 6.2)

问题 6.3 解答略(参考课文)。

问题 6.4 解答略(参照 4.2 节的(b)～(d)。用适当的满足 c0<c2 的分频分母 c0 和 c2 确定)。

问题 6.5 4 位的数值用 16 进制表示是 0h～Fh，或用 10 进制表示为 0～15，共 16 个。模拟电压的范围为 8V。因此，数字量的 1 步(根据式(6.1)所对应的电压为：

$$\frac{8V}{16}=0.5V$$

于是，如图 6.A 所示，数字量的 8 对应 0V，用 0～8 的 8 个段对应 −4V，8～15 的 7 个阶段对应(4-0.5)V。

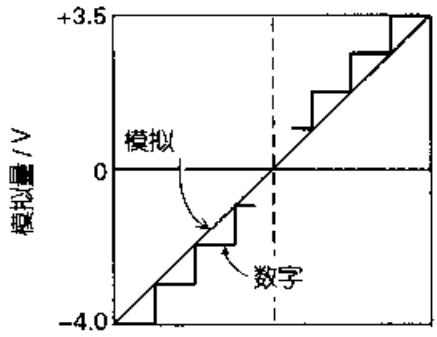


图 6.A　AD 转换示例

问题 6.6　因 0 对应-5V、2048 对应 0V，所以 −2.5V 对应 1024。同样地，因 2048 对应 0V、4096 对应 +5V，所以 2048 + 1024 = 3072 对应 +2.5V。函数 walt 要在 1s 间输出从 1024 到 3072 的等间隔的 1000 个数据，每个的时间是 1ms。

❖练习 7

问题 7.1　关于 C 的行内汇编可参照 3.3 节(b)。

```
#include <stdio.h>
#include <dos.h>
void main( )
{
    printf("\x1b" "[2J");      /* 清视频 */
    _asm
    {
      mov ah,     02
      mov bh,     0
      mov dh,     5
      mov dl,     10          /* 与 Turbo-C 的 gotoxy(x, y)函数相当 */
      int 10h
    }
}
```

问题 7.2　以下代码在控制台输出字符"a"。

```
#include <stdio.h>
```

```
#include <dos.h>
union REGS inregs, outregs;
int out_chara(int func, char ch)
{
    int d0 = 0;
    if(func == 2 || func == 4 || func == 5 || func == 6)
    {
      inregs.h.ah = func;
      inregs.h.al = ch;
      intdos(&inregs, & outregs);
    }
    else
      d0 = -1;
    return d0;
}
void main( )
{
    int ah = 2;
    char cha = 0x61;
    out_chara(ah, cha);
}
```

问题 7.3 有以下 3 种方法：

①B:\>copy b:\bx\bcd.doc b:\cx\cde.doc
  B:\>cd bx
②B:\BX>copy bcd.doc b:\cx\cde.doc
③B:\>c:
  C:\>cd cx
  C:\CX>copy a:\bx\acd.doc cde.doc

问题 7.4 为了判别是否有按键，事先需要清除键盘缓冲区。行内汇编代码如下(请自行修改为 C 语言代码)：

```
void clear_key(void)
{
    _asm
    {
      r00:    movah, 1h ;key sense 可参考表 7.3
              int 16h
              jnz r10
              jmp end
      r10:          ;有键按下
```

```
            movah, 00h
            Int 16h
            jmp r00
        }
    }
```

问题 7.5

```
// sense receive data, then display
// sense key Input, then send
# include <stdlo.h>
# include <stdlib.h>
# include <dos.h>
# include "rs232c.h"
void main( )
{
    int stat, parameter, port;
    char charac, key;
    port = 1;
    parameter = 0xe3;              // 0xe3 = 9600 bps, non - parity,
stop 1, 8 bit
    stat = init_rs232c(port, parameter);

    while(1)
    {   // receive
      If((status_rs232c(port) & 0x100) = = 0x100)
                    // check receive data
      {
      printf("\nreceivable data exist");
      stat = receive_rs232c(port, &charac);
      printf("\n%c", charac);
      printf("\nreceive status %x", stat);
      }
      else
      printf("\nreceivable data none");
      // ESC key
      If(kbhit( ) ! = 0)
      {
      key = getch( );
      if(key = = 0xb)
        break;              // ESC
      // send
      if((status_rs232c(port) & 0x2000) = = 0x2000)
```

```
        {
          printf("\nsend ready");
          stat = send_rs232c(port, key);      //send
          printf("\nsend status %x", stat);
          if((stat & 0x80) = = 0x80)
          {
            printf("\ntime out er!");  exit(1);
          }
        }
        else
          printf("\nsend none");
        }
      }
}
```

❖练习 8

问题 8.1 编译仅包含定义于头文件 dlscon.h 中的函数 clrscr( )的 C 语言程序，形成可执行文件 clrscr.exe，存放在原文件所在的文件夹。printf("\x1b" "[2J");。

问题 8.2 模式 72h。以下分别为使用 BIOS 和 ESC 序列进行模式设置。

```
union REGS inregs, outregs;
void set_videomode(int mod)
{
    inregs.h.ah = 0;
    inregs.h.al = mod;
    int86(0x10, &inregs, &outregs);
}
```

以及

```
void set_videomode(int mod)
{
    printf("\x1b" "%dh", mod);
}
```

问题 8.3

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <graph.h>                          /* 图形头文件 */
void main(void)
{
    _setvideomode(VRES16EXCOLOR);
    _setviewport(0, 0, 639, 479/2);         /* 物理屏幕的上半部分 */
```

```
    clearscreen(GCLEARSCREEN);          /* 清除画面 */
    _setwindow(1, -50., -100., 50., 100.);    /* 用户窗口 */
                    /* 1是左下角最小值,右上角最大值 */
    movetow(px0, py0);                  /* 画线 */
    linetow(px1, py1);
}
```

问题 8.4

```
void mouse_getpos(int * right, int * left, int * x, int * y)
{
    inregs.x.ax = 3;
    int86(0x33, &inregs, &outregs);
    * left = outregs.x.bx & 0x01;       /* left button */
    * right = outregs.x.bx & 0x02;      /* right button */
    /* 这里插入判断处理代码 */
    * x = outregs.x.cx;                 /* x axis */
    * y = outregs.x.dx;                 /* y axis */
}
```

问题 8.5 解答略(提示:函数参数为 1 时对应 ON,为 0 时对应 OFF)。

❖练习 9

问题 9.1

```
void mcw(int count)
{
    int i;
    for(i=0; i<count; i++)
    {
      outp(PB, 0x09);       wait( );
      outp(PB, 0x0a);       wait( );
      outp(PB, 0x06);       wait( );
      outp(PB, 0x05);       wait( );
    }
}
```

问题 9.2 此步进电机经 100 个脉冲转动 1 周。这里只给出了主函数部分。

```
#include <stdio.h>
#include <dos.h>
void main(void)
{
    char cc[10]; int i;
    int count = 100;            /* 转动 1 周计数 */
    outp(CR, CW);               /* 8255A 的初始化 */
    outp(PB, CWP0);             /* 电机启动准备 */
```

```
    gets(cc);                    /* 等待启动 */
    for(i=0; i<count; i++)
      mcw(PB, 250);              /* 正转 */
    moff(PB);
    while(kbhit( ))              /* 若有键盘输入进行下一步 */
    {
      for(i=0; i<count; i++)
      mccw(PB, 250);
    }
    moff(PB);                    /* 停止 */
}
```

问题 9.3 $q_0 = 500, q_r = 1500, n = 10$。根据式(9.3),有:

$$\alpha = \frac{2 \times (1500^2 - 500^2)}{\sqrt{17^2 + 3^2 - 1 + 17}} = 116.844 \text{ step/s}^2$$

$$g = 383.2 \text{ pps}$$

因此,脉冲计时t_i为:

$$t_i = \frac{\sqrt{383.2^2 + 2 \times 116.844 \times i} - 383.2}{116.844}$$

而在定常状态时有:

$$t_i = \frac{1}{1500}$$

问题 9.4 长处包括:①转矩与电流成正比,可控性强。②启动转矩大。③转数与端子电压成正比,精度高,动作幅度大。短处包括:④必须保留电刷和换向器。⑤电火花是导致电磁干扰的原因。

参考文献

[1] 松下電器工科短大 編：ロボットⅠ・マイコン応用技術，廣済堂出版（1992）
[2] 柳田博明：センサ先端技術，海文堂（1986）
[3] 山崎弘郎，森村正直 編：センサエ学，朝倉書店（1990）
[4] 特集「オプト・デバイス応用回路設計・製作」，トランジスタ技術スペシャル，No. 33，CQ出版（1992）
[5] 綿貫啓一：メカトロニクスシステムの設計・制御技法，日刊工業新聞社（1995）
[6] 藤田英時：DOS/Vの素朴な疑問、日本実業出版社（1996）
[7] 谷腰欣司：メカトロニクスのためのセンサ応用回路101選,日刊工業新聞社（1991）
[8] 横山直隆：C言語による製作と制御実習入門，シータスク（1996）
[9] 松下電器工科短大 編：マイコン，廣済堂出版（1991）
[10] 白土義男：マイコン回路の手ほどき，日本放送出版協会（1989）
[11] MP-85mkII用インタフェース，サン・マイテック社（1993）
[12] 武藤 一夫：メカトロニクスとマイコンⅠ・II，工学図書（1995）
[13] Microsoft：Microsoft C Programming Guide （1991）
[14] 特集「パソコンで広がる電子工作の世界」，トランジスタ技術，CQ出版（1996.8）
[15] 吉野敏也：Windowsに隠されたDOSの秘密，ナツメ社（1996）
[16] 横山直隆：パソコン機械制御と製作実習入門，技術評論社（1990）
[17] 相沢一石：8086ファミリ・ハンドブック，CQ出版（1991）
[18] 太平洋工業 編：PC-9801パソコンによる機械制御実習，日刊工業新聞社（1979）
[19] 天良和男：はじめてのパソコン計測・制御，東京電機大学出版局（1989）
[20] 割込みプログラミング技法，別冊インタフェース，CQ出版（1995）
[21] 鈴木美朗志：ポケコン制御実習，オーム社（1999）
[22] 割込みプログラミング技法，別冊インタフェース，CQ出版（1995）
[23] 渡辺郁朗：PCユーザの基礎知識・ＡＴ互換機のバイオス、工・アイ出版（1998）
[24] 服部昌博：DOS/V・BIOSとC言語，工学図書（1996）
[25] アスキーテクライト編：PC-98シリーズ・テクニカルデータブック，アスキー出版局(1992)
[26] 磯部俊夫：C言語と割込み，工学図書（1990）
[27] サン・マイテックYV10取扱説明書，サン・マイテック社（1998）
[28] 杉浦朋美：DOS/Vプログラミング技法，翔泳社（1994）
[29] 服部昌博：DOS/VとC言語，工学図書（1994）
[30] 横山直隆：パソコン・インタフェースの製作実習，技術評論社（1986）
[31] 津田利春：PC-98拡張ボードの基本，工学図書（1992）
[32] ユーザガイド編：ANALOG-PRO I/II USER' S GUIDE，カノープス社（1995）
[33] 綿貫啓 一：メカトロニクスシステムの設計・制御技法，日刊工業新聞社（1995）
[34] IBX 3148取扱説明書，インタフェース社（1998）
[35] ADM-640 AT取扱説明書，マイクロサイエンス社（1998）

[36] DA12-4（PC)解説書，コンテック社（1997）
[37] 川上峻史：BIOSとCプログラミング，工学図書（1991）
[38] 守口俊伸：とことんこだわるCONFIG.SYS，工一アイ出版（1993)
[39] 工一アイムック44「config.sys&メモリのすべて」，工一アイ出版（1993)
[40] 村瀬康治：応用MS-DOS，アスキー出版（1990）
[41] 渡辺嘉二郎：パソコンによるセンサ信号処理，海文堂（1991）
[42] 川上峻史：キーボード・マウス入力とC言語，工学図書（1992）
[43] 真壁國昭：ステッピングモータの制御回路設計，CQ出版（1991)
[44] 加藤 一：小形モータ制御用IC，工業調査会（1988)
[45] 見城尚志：アセンブリ/Cによるメカトロ制御，総合電子出版社（1990）
[46] General Catalog，コパル電子株式会社（1995)
[47] 4軸ステッピングモータ・コントローラUser's Manual，インタフェース社（1995)
[48] 見城尚志，永守重信：メカトロニクスのためのDCサーボモータ，総合電子出版社（1983）
[49] 牧野 洋：自動機械機構学，日刊工業新聞社（1984）
[50] 山藤和男：テクノライフ選書 ロボットAto Z，オーム社（1997）

[36] DA12-4（PC)解説書，コンテック社（1997）
[37] 川上峻史：BIOSとCプログラミング，工学図書（1991）
[38] 守口俊伸：とことんこだわるCONFIG.SYS，工一アイ出版（1993)
[39] 工一アイムック44「config.sys&メモリのすべて」，工一アイ出版（1993)
[40] 村瀬康治：応用MS-DOS，アスキー出版（1990）
[41] 渡辺嘉二郎：パソコンによるセンサ信号処理，海文堂（1991）
[42] 川上峻史：キーボード・マウス入力とC言語，工学図書（1992）
[43] 真壁國昭：ステッピングモータの制御回路設計，CQ出版（1991)
[44] 加藤 一：小形モータ制御用IC，工業調査会（1988)
[45] 見城尚志：アセンブリ/Cによるメカトロ制御，総合電子出版社（1990）
[46] General Catalog，コパル電子株式会社（1995)
[47] 4軸ステッピングモータ・コントローラUser's Manual，インタフェース社（1995)
[48] 見城尚志，永守重信：メカトロニクスのためのDCサーボモータ，総合電子出版社（1983）
[49] 牧野 洋：自動機械機構学，日刊工業新聞社（1984）
[50] 山藤和男：テクノライフ選書 ロボットAto Z，オーム社（1997）